Component Relationships Depend on Skill in Programming?

Jeffrey S. Feddon and Neil Charness

Department of Psychology
Florida State University
Tallahassee, FL 32306
Phone: 850-644-9869
feddon@psy.fsu.edu
http://www.psy.fsu.edu/~feddon/homepage.htm

Paper presented at the 11th Annual PPIG Workshop, University of Leeds, UK, January 4-7, 1999.

Introduction

Computer programming studies commonly view programming either as an aggregate task or as a set of components (subtasks). Shneiderman and Mayer (1979) and Brooks (1977) were early attempts to model general cognitive processes in programming. The complexity of examining programming as a single problem-solving task soon became obvious and researchers began following early suggestions (e.g., Shneiderman, 1976) in focussing intently on individual subtasks. Highly detailed accounts of programmer behavior now exist in the literature for some subtasks, including software design (Jeffries, Turner, Polson, and Atwood, 1981), comprehension (Brooks, 1983), and debugging (Vessey, 1989).

For background purposes, the following are examples (with brief definitions) of common subtasks in programming (Shneiderman, 1976; Koubek, Salvendy, Dunsmore, & Lebold, 1989): (1) Software Design : deciding problem requirements, designing an optimal solution to the problem, and creating a mental representation of the procedure; (2) Comprehension: understanding a program--what does a program do?; (3) Composition: translating a representation to program code; (4) Debugging: determining why a program does not function or functions improperly; and (5) Modification: making changes to an existing functional program.

Koubek et al. (1989) argue that subtasks required in programming are distinct, but interrelated. Unfortunately, very little evidence exists about the nature of these interrelationships (Bishop-Clark, 1995; Pennington, Nicolich, & Rahm, 1995). The most extensive work comes from studies examining the extent and type (declarative vs. procedural) of knowledge transfer between evaluation (comprehension) and generation (composition) of LISP procedures (Anderson, Conrad, & Corbett, 1989; Pennington et al., 1995). However, this research is in its infancy and the results are inconclusive. Further, these studies did not examine the impact of skill on subtask performance (a focus of the present study).

Some have hypothesized that certain subtasks share common or similar cognitive components (e.g., Bishop-Clark, 1995; Koubek et al., 1989; Shneiderman & Mayer, 1979). For example, Bishop-Clark (1995) says that program debugging requires comprehension. Koubek et al. (1989) say that "modification requires a programmer to use the combined skills of comprehension, composition, and debugging" (p. 183). Shneiderman and Mayer (1979) state that program comprehension requires subtasks of debugging, modification, and learning. Mynatt (1984) makes the claim that "whether writing, debugging, or modifying software, comprehension is involved" (p. 91). While most of these statements are logical assertions, not much concrete evidence exists about the extent of subtask overlap.

Moher and Schneider (1982) suggested that understanding the correlation between performance on various subtasks (including the impact of skill) is an important line of future research. Ideas for the present study follow from this suggestion and the need for further research on subtask relationships. The present study sought to investigate the degree of relationship between subtasks of comprehension, debugging, composition, and modification. Subjects were college students who had knowledge of the C programming language. They were designated to one of three possible skill categories (novice, intermediate, and advanced). All subjects solved two programming problems for each subtask.

Statistical analyses consisted of examining relationships between subtasks across skill groups and within each skill group. Further, we examined mean differences between skill groups for each subtask measure and for subtask measures combined. Because this was an exploratory study, researchers made no specific hypotheses beyond general assumptions that subtask relationships would change with skill and that higher-skilled programmers would do better than lower-skilled ones.

Method

Subjects

Subjects were 35 males and females in three possible skill conditions (novice, intermediate, advanced). Novices consisted of 11 persons who had just finished a semester course on C programming and had no previous programming experience other than this course. There were 23 Intermediate subjects. Nine of these subjects had just finished the same C programming course, but had programming experience before the course. The other 12 intermediates had previously taken a course on C programming and had other programming experience. There were three advanced subjects and all were computer science graduate students, two of which had previously taught a course on C programming.

Stimuli and Apparatus

The stimuli consisted of two test forms (A and B) counterbalanced across skill categories. Each test form contained eight programming problems (two problems for each of the four subtasks). There were 16 total problems (4 for each subtask) and no problems were the same across test forms. The researchers developed some problems and selected others from example problems in college-level C programming books (e.g., Johnsonbaugh & Kalin, 1993). Comprehension, debugging, and modification problems were less than 30 lines long and composition problems required less than 30 lines of code. All problems were complete programs (i.e., not code fragments).

To combat possible range effects (e.g., ceiling effects due to easy problems), for all subtasks, researchers developed problems that ranged in difficulty. Debugging problem difficulty was based on the number of bugs in the program (problems contained either 5, 4, 3, or 2 bugs). Researchers assessed problem difficulty for comprehension, composition, and modification problems by two methods. Initially, researchers used a rating system to assess difficulty level. The rating system allocated a varying number of points based on the complexity of common programming structures. The five structure categories (with point values) were: (1) A looping structure (five points); (2) An IF statement (three points); (3) An array (two points); (4) A variable (one point); and (5) All other statements (one- half point for each statement). Problems with higher scores were considered more difficult.

An experienced C programmer concurrently validated rating system scores. Researchers asked the programmer (for each subtask) to sort the four problems on the basis of difficulty: one for most difficult, two for second, three for third, and four for least difficult. Based on agreement between rating system scores and sort orders, the researchers and the experienced programmer worked together in reaching a consensus about difficulty ratings. We developed test forms A and B in the following manner. Form A

contained comprehension problems of difficulty levels 2 and 3, debugging (level 1 = 5 bugs and level 4 = 2 bugs), composition (levels 2 and 3), and modification (levels 1 and 4). Form B contained the remaining problems and is completely orthogonal to form A.

Researchers developed four practice problems (one of each subtask type) to instruct subjects on the experimental tasks and to control for practice effects. We used the same problem (with minor modifications) as an example for each subtask.

Researchers wrote a program that controlled problem display. The program interfaced with the Turbo C++ 3.0 for DOS compiler IDE. Problems were displayed in the IDE to maintain an ecologically-valid setting and to allow subjects to have facilities for compiling and debugging programs.

Design

The design used in this experiment was a one-way between subjects multivariate design. Skill level was the independent variable with three conditions (novice, intermediate, and advanced). The four dependent variables were scores on each of the four subtask measures. Debugging scores were based on the number of bugs corrected (seven points possible for each subject). Researchers developed and used a rating scheme (based on a zero to five scale) to score performance in comprehension (based on completeness of description), composition, and modification (both based on completeness of solution). Problems were scored: "5" if a problem was completely correct or had one minor syntax error; "4" if there was one major (semantic) or two minor (syntactic) errors; "3" if there were two major errors, one major and one or two minor errors, or three or four minor errors; "2" if there were no more than three major errors; "1" if the problem was partially correct and contained more than three major errors; and "0" if the problem solution was insufficient, wrong, or not stated.

We used random counterbalancing to control for order effects. The computer randomly generated problem orders for each subject. Therefore, each subject was equally likely to receive problems in any one of the possible permutations for eight conditions.

Procedure

Subjects signed an informed consent form and read a general description of the experiment. Subjects were then given a brief introduction to the Turbo C++ IDE. The experimenter informed them about things like how to expand windows, scroll up and down, compile code, save their file, and get to the next problem. After subjects said they understood what to do and how to use the IDE, they proceeded to a practice problem phase. The experimenter guided subjects through the practice problems (one for each subtask) and answered any questions. For comprehension problems, researchers instructed subjects to describe in their own words what the program does. Subjects were instructed to locate and fix bugs in debugging problems. For composition problems, subjects were instructed to write a program that meets stated specifications. For modification problems, subjects were given a program with specifications for current functionality and provided with additional specifications for modifying the program.

After practice, the experimenter told subjects that they should attempt to solve all eight programming problems, providing a best possible solution to each problem. The experimenter told subjects that they had complete control over their rate of progress, but that they should work as quickly as possible. However, the experimenter emphasized accuracy (good solution) over speed (fast solution). Subjects were told that if they had extreme difficulties with a problem (e.g., felt they could progress no further), they should move on to the next problem. Subjects could take breaks (at their leisure) between problems. The experiment ended when subjects finished all problems. At this point, the experimenter debriefed subjects and answered any questions.

Results

We scored problems (using the above rating scheme) by comparing each problem to an idealized solution. Two persons (one researcher and a volunteer) scored problems. The researcher scored all problems (these scores were used for data analysis), while the volunteer scored a sample of problems (one novice--form A, one novice--form B, two intermediates--form A, two intermediates--form B, one advanced--form A, and one advanced--form B). A significant Pearson correlation ($r = .76$, $p < .001$) showed modest inter-rater agreement across all subtask problems. A look at inter-rater agreement by subtask shows that while agreement was high for debugging ($r = .94$, $p < .001$) and composition ($r = .89$, $p < .001$) problems, agreement was lesser for modification ($r = .71$, $p < .002$) and not very good for comprehension ($r = .41$, ns) problems. A probable cause for such low agreement for comprehension problems is a lack of sensitivity of the rating scheme. While the scale is more objective for problems containing "program code" responses, it is more susceptible to subjective interpretation when subjects were to provide English language descriptions for comprehension problems.

The experimenters aggregated problem scores in arriving at a subtask score for each subject. Comprehension, composition, and modification scores were obtained by adding results of the two problems for each subtask. Given that five was a maximum score for each problem, subtask scores are based on the number of points out of 10. The two debugging problems (for both test forms) contained a total of seven bugs and scores were first based on the number of bugs corrected out of seven. The experimenters then converted debugging scores to a 10-point scale by the following computation [Debugging Score = ((number of bugs corrected) / 7) * 10]. Therefore, there were 40 total points possible across the four subtasks.

We checked test forms A and B for equivalence by examining total score averages and subtask score averages between the forms. The total score means for form A (26.65) and form B (27.46) were very similar and not significantly different. No significant differences were found between form A and B subtask score means (Comprehension - 7.74 (A), 7.81 (B); Debugging - 6.54, 6.96; Composition - 5.84, 6.81; Modification - 6.53, 5.88). These results provide confidence that the test forms were equivalent.

The following figures provide descriptive statistics information. Table 1 shows the means and standard deviations for each skill level and subtask, including totals across each variable. Figure 1 is a line graph of the subtask score averages for each skill level. Figures 2 is a stacked bar graph of mean subtask scores and shows the aggregate mean for each skill level.

One-way ANOVAs examined total (aggregate) score average and subtask score averages between skill levels. The ANOVA for total score average [$F(2, 32) = 4.51$, $p = .019$] was significant. The ANOVAs for debugging [$F(2, 32) = 3.15$, $p = .056$], composition [$F(2, 32) = 3.09$, $p = .059$], and modification [$F(2, 32) = 2.90$, $p = .070$] were marginally significant, while that for comprehension was not significant [$F(2, 32) = .69$, $p = .511$].

Tukey post hoc tests were done on the significant and marginally significant ANOVAs. For total score, there was a significant difference between novice (22.75) and advanced (33.71) group means, but neither differentiated from the intermediate group (28.30). For subtasks, debugging score showed the only significant pairwise difference between novice (6.30) and advanced (9.05) groups.

Table 2 shows the overall (across skill level) correlations between subtasks. Notice that the correlations between composition and modification ($r = .61$, $p < .001$) and between comprehension and composition ($r = .35$, $p = .039$) were significant. Tables 3, 4, and 5 show correlations between subtasks broken down by skill level (novice, intermediate, and advanced). For novice programmers, while no correlations were significant, the strongest correlations were between comprehension and composition ($r = .47$) and composition and modification ($r = .39$). For intermediate programmers, the correlation between composition and modification was significant ($r = .69$, $p < .001$) and no other correlations were

significant or very strong. Given that there were only three subjects in the advanced condition, correlations are not worthy of discussion.

A scatterplot of skill level and total score reveals extensive individual differences within novice and intermediate groups (range = 22.14 and 23.29, respectively), suggesting that subject' skill classification based on a broad measure of experience is not very robust (cf. Collani & Schömann, 1995). While the correlation was significant ($r = .47$, $p = .004$), skill categories captured only 22% of the variance in total scores (see Figure 3).

While acknowledging sample size limitations, a final examination of the data entailed doing an exploratory factor analysis (principal component analysis with varimax rotation) for the four subtasks across all subjects. Two distinct factors were extracted with Eigenvalues of 1.926 and 1.036, accounting for 46% and 26% (~72%) of the total variance, respectively. Composition (.843) and modification (.914) loaded heavily on the first factor, while comprehension (.713) and debugging (.863) loaded heavily on the second factor.

For a finer-grained inspection, factor analyses were done separately for novice and intermediate groups. For the novice group, two factors were extracted with Eigenvalues of 1.884 and 1.084, accounting for 47% and 27% of the variance. Comprehension (.768), composition (.776), and modification (.795) loaded heavily on the first factor, while debugging (.940) loaded heavily and exclusively on the second factor. For the intermediate group, again two factors were extracted (Eigenvalues of 1.806 and 1.213--accounting for 45% and 30% of the variance), but composition (.895) and modification (.932) loaded heavily on the first factor, while comprehension (.798) and debugging (.791) loaded heavily on the second factor.

Discussion

Providing some exploratory evidence about the relationships between subtasks and differences between subtask performance based on skill level, were the major goals of this research. At the outset, we wish to note that sample size in the advanced group is a severe limitation for data analysis. However, this research is still in progress and we plan to get more novice and advanced subjects to equalize sample sizes. A further limitation is the insensitivity or lack of objectivity of certain aspects of the rating system used with some subtasks (e.g., comprehension). While acknowledging these problems, we wish to focus here on examining and interpreting the data.

At the present stage of the research, results suggest that relationships between subtasks depend on skill level and that some subtasks (e.g., composition and modification) are more strongly related than others. Further, there is some evidence to support hypotheses discussed earlier. For example, factor analysis results support Bishop-Clark's (1995) statement that program debugging requires comprehension. However, results suggest that this may be skill-dependent, as the intermediate group results provide stronger evidence than the novice group.

It may be useful to offer an interpretation for the two factors extracted in the overall factor analysis. At least two questions are relevant: (1) What underlying process(es) might composition and modification use that comprehension and debugging do not and vice versa?; and (2) Are these factors sensitive to skill differences? By definition, modifying a program requires one to do some coding (composition). Alternatively, both comprehension and debugging require one to inspect/scan a program, to build an understanding and to locate problems. Possibly, factor one is related to generation processes and factor two involves evaluation processes (cf. Anderson et al., 1989; Pennington et al., 1995). To address the second question we examined the correlation between each factor and skill level. The correlation between factor one and skill level was significant ($r = .38$, $p = .024$), while factor two did not significantly correlate with skill level ($r = .27$). A tentative interpretation is that generation processes

may be more skill dependent than evaluation processes, but further work is needed to increase confidence in such claims.

While ANOVA examinations of mean differences did not reveal a great deal, graphs show a trend of increased performance with separate subtasks (Figure 1) and for subtasks aggregated (Figure 2). Besides small samples, insensitivity from extensive individual differences for novice and intermediate groups may explain few significant mean differences (see Figure 3). Apparently, large individual differences are common in computer programming studies. Moher and Schneider (1982) suggest that, "in several reports, it has been noted that variability due to subject differences often outweighs variability due to independent variables...to date, the classification of subjects has generally been based on very rough measures, usually relating to academic backgrounds" (p. 73).

Future Directions

Our current work involves plans of formulating methods of predicting and/or extracting out extensive variability from individual differences. Specifically, we are developing a theory to describe the underlying structures used in programming that includes specific predictions about the circumstances under which one would expect relationships to occur between subtasks and how these relationships might be affected by an individual's basic capacity (hardware) and accumulated knowledge structures (software). At this juncture, we will be in a better position to address the goals of the present exploration more critically. Plans are to develop studies that enable us to work in parallel on these issues.

References

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, 13, 467-506.

Bishop-Clark, C. (1995). Cognitive style, personality, and computer programming. *Computers in Human Behavior*, 11, 241-260.

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man--Machine Studies*, 9, 737-751.

Collani, G. V., & Schömann, M. (1995). The process of acquisition of a new programming language (LISP): Evidence for transfer of experience and knowledge in programming. In K. F. Wender, F. Schmalhofer, and H. D. Böcker (Eds.), *Cognition and computer programming* (pp. 169-191). Norwood, NJ: Ablex Publishing Corporation.

Jeffries, R., Turner, A. A., Polson, P. G., & Atwood, M. E. (1981). The processes involved in designing software. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition* (pp. 255-283). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Johnsonbaugh, R., & Kalin, M. (1993). *Applications programming in ANSI C* (2nd ed.). NY: Macmillan Publishing Company.

Koubek, R. J., Salvendy, G., Dunsmore, H. E., & LeBold, W. K. (1989). Cognitive issues in the process of software development: review and reappraisal. *International Journal of Man--Machine Studies*, 30, 171-191.

Moher, T., & Schneider, G. M. (1982). Methodology and experimental research in software engineering. *International Journal of Man--Machine Studies*, 16, 65-87.

Mynatt, B. T. (1984). The effect of semantic complexity on the comprehension of program modules. *International Journal of Man-Machine Studies*, 21, 91-103.

Shneiderman, B. (1976). Exploratory experiments in programmer behavior. *International Journal of Computer and Information Sciences*, 5, 123-143.

Shneiderman, B., & Mayer, R. (1979). Syntactic/Semantic interactions in programmer behavior: A model and experimental results. *International Journal of Computer and Information Sciences*, 8, 219-238.

Vessey, I (1989). Toward a theory of computer bugs: An empirical test. *International Journal of Man-Machine Studies*, 30, 23-46.

Wiedenbeck, S. (1986). Beacons in computer program comprehension. *International Journal of Man-Machine Studies*, 25, 697-709.

Table 1 - Means and Standard Deviations

Skill Level by Subtask

Skill		Comprehension	Debugging	Composition	Modification	Total
Novice	Mean	7.09	6.30	4.64	4.73	22.75
	N	11	11	11	11	11
	Std. Deviation	2.81	2.04	2.77	2.90	7.20
Intermediate	Mean	8.05	6.63	6.81	6.81	28.30
	N	21	21	21	21	21
	Std. Deviation	2.13	1.58	3.04	2.38	6.28
Advanced	Mean	8.33	9.05	8.67	7.67	33.71
	N	3	3	3	3	3
	Std. Deviation	2.08	.82	2.31	2.52	1.03
Total	Mean	7.77	6.73	6.29	6.23	27.02
	N	35	35	35	35	35
	Std. Deviation	2.34	1.81	3.09	2.70	7.02

Figure 1 - Subtask Means

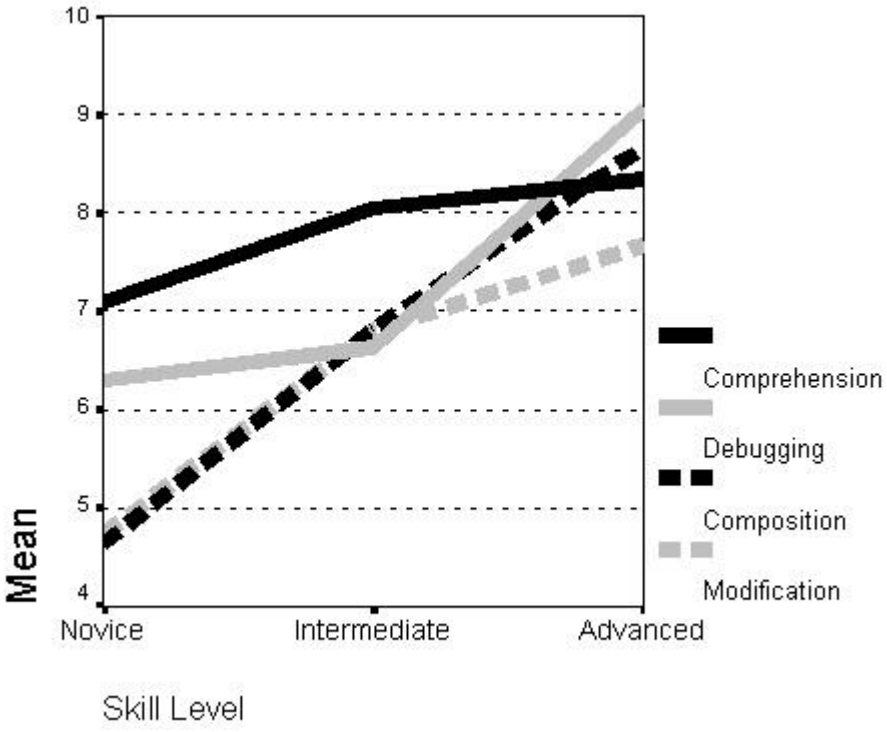


Figure 2 - Subtask and Aggregate Means

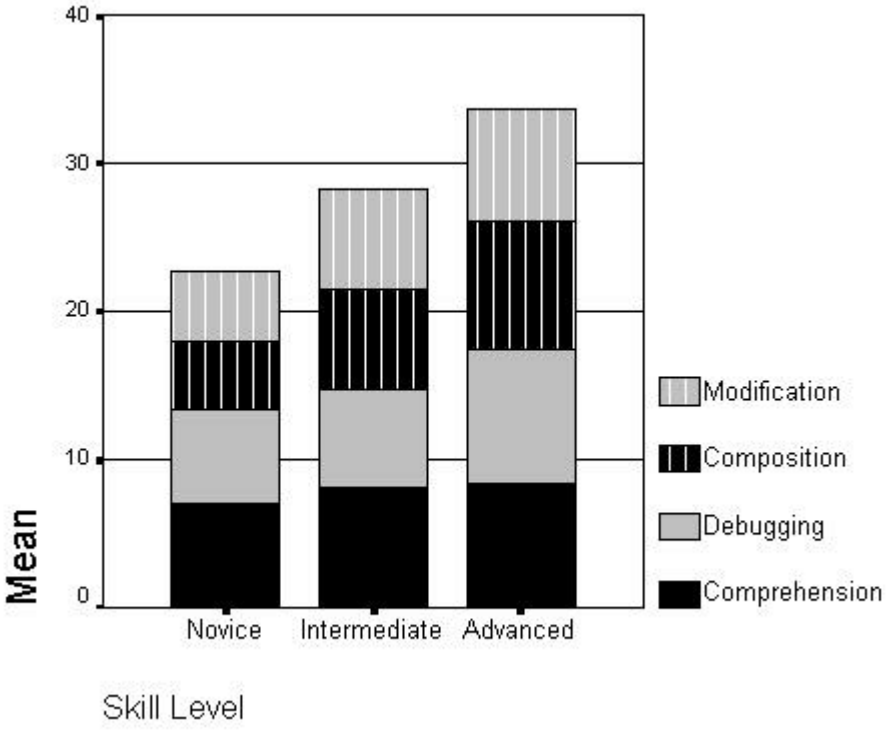


Table 2 - Total Correlations

Overall Correlations (N = 35)

		Comprehension	Debugging	Composition	Modification	CLevel
Pearson Correlation	Comprehension	1.000	.300	.351*	.204	.193
	Debugging	.300	1.000	.239	.071	.319*
	Composition	.351*	.239	1.000	.608**	.402*
	Modification	.204	.071	.608**	1.000	.379*
	Skill Level	.193	.319*	.402*	.379*	1.000
Sig. (2-tailed)	Comprehension	.	.080	.039	.240	.267
	Debugging	.080	.	.167	.685	.062
	Composition	.039	.167	.	.000	.017
	Modification	.240	.685	.000	.	.025
	Skill Level	.267	.062	.017	.025	.

*. p < .05

**. p < .001

Table 3 - Novice Correlations

Novice Correlations (N = 11)

		Comprehension	Debugging	Composition	Modification
Pearson Correlation	Comprehension	1.000	.264	.468	.396
	Debugging	.264	1.000	.168	-.091
	Composition	.468	.168	1.000	.385
	Modification	.396	-.091	.385	1.000
Sig. (2-tailed)	Comprehension	.	.433	.147	.228
	Debugging	.433	.	.622	.790
	Composition	.147	.622	.	.242
	Modification	.228	.790	.242	.

Table 4 - Intermediate Correlations

Intermediate Correlations (N = 21)

		Comprehension	Debugging	Composition	Modification
Pearson Correlation	Comprehension	1.000	.283	.287	.012
	Debugging	.283	1.000	.172	.011
	Composition	.287	.172	1.000	.685**
	Modification	.012	.011	.685**	1.000
Sig. (2-tailed)	Comprehension	.	.215	.208	.960
	Debugging	.215	.	.457	.963
	Composition	.208	.457	.	.001
	Modification	.960	.963	.001	.

**. p < .001

Table 5 - Advanced Correlations

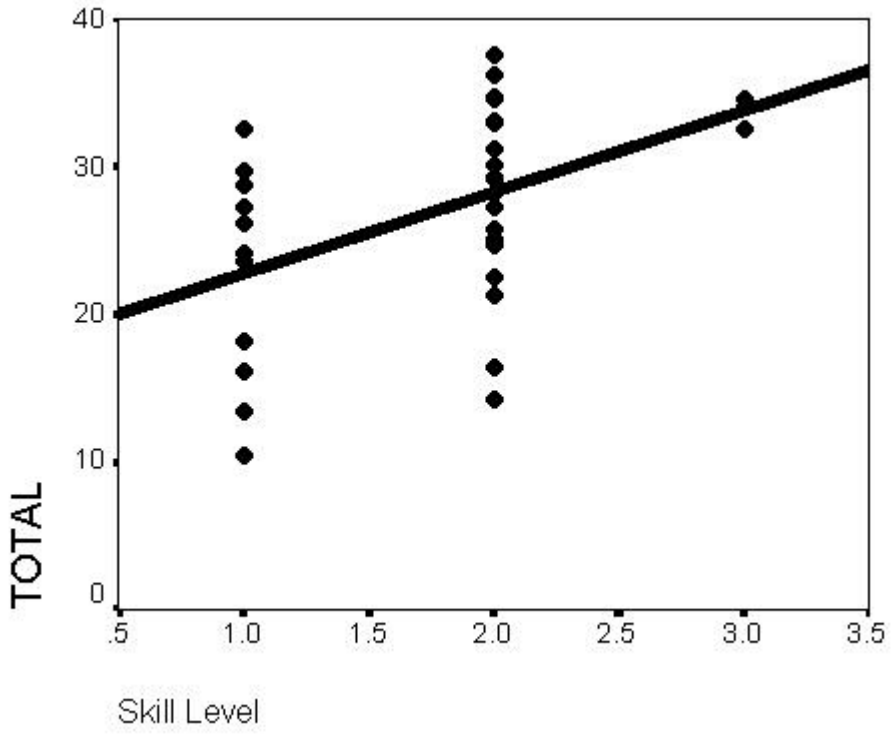
Advanced Correlations (N = 3)

		Comprehension	Debugging	Composition	Modification
Pearson Correlation	Comprehension	1.000	.693	-.693	-.636
	Debugging	.693	1.000	-1.000**	.115
	Composition	-.693	-1.000**	1.000	-.115
	Modification	-.636	.115	-.115	1.000
Sig. (2-tailed)	Comprehension	.	.512	.512	.561
	Debugging	.512	.	.000	.927
	Composition	.512	.000	.	.927
	Modification	.561	.927	.927	.

**. p < .001

Figure 3 - Skill Level by Total -- Rsquare = .22

1 = Novice, 2 = Intermediate, 3 = Adanced


40
30
20
10
0
TOTAL
.5
1.0
1.5
2.0
2.5
3.0
3.5
Skill Level