EUC TRAINING: COMPARISON OF METHODS AND THE ROLE OF INDIVIDUAL DIFFERENCES

Lorne Olfman
Programs in Information Sciences
The Claremont Graduate School
Claremont, California 91711
(OLFMANL@CLARGRAD)

Maung K. Sein
Department of Decision Sciences and Information Systems
College of Business Administration
Florida International University
Miami, Florida 33199
(SEINM@SERVAX)

Robert P. Bostrom
Department of Management
College of Business Administration
University of Georgia
Athens, Georgia 30602
(BBOSTROM@UGA)

ABSTRACT

The importance of effective end-user training to the organization is being recognized by researchers in end-user software training. The research reported here is based on a study that compared two seminar/classroom software training methods in terms of understanding outcomes. Applications-based training was designed to be more personally relevant than construct-based training, which was designed to be similar to traditional methods of training. The expectation was that overall understanding outcomes would be higher for trainees who received construct-based training. However, for some trainees, depending on individual differences in previous computer experience and cognitive abilities, applications-based training was expected to enhance training outcomes. Understanding of the target software, IFPS, was measured in terms of three dimensions: architecture, function, and syntax. Results showed that for function understanding, construct-based training was expected to enhance training outcomes. Understanding of the target software, IFPS, was measured in terms of three dimensions: architecture, function, and syntax. Results showed that for function understanding, construct-based training was superior. For syntax understanding, those with high previous experience and low "crystallized" intelligence had the highest levels of understanding. There were also some near significant interactions between training methods and individual differences. Suggested directions for researchers and implications for practitioners are discussed.

1. INTRODUCTION

End-user training is a critical factor that must be considered by information systems (IS) managers and researchers (Cheney, Mann and Amoroso [6]). End-user computing (EUC) transfers full responsibility for development of a particular system into the hands of users. However, the overall management and facilitation of end-user development must continue to be a function of the IS department. IS is responsible for strategic system planning and has experience and knowledge in systems development. The emerging role for IS in terms of EUC should therefore include training. Effective training can provide the end-user with an accurate initial understanding and motivation to use the computer software that she/he requires to build her/his own effective systems.

The importance of end-user training has not been reflected in empirically-tested guidelines for designing effective training methods. At least two reasons can be postulated for this situation. First, it is generally claimed by software vendors and some researchers that end-user software packages are easy to learn and use. However, there is much evidence to counter these claims. Studies concerning the learning of end-user software have consistently shown that learning these software packages is not easy (Mack, Lewis and Carroll [17]; Carroll [3]; Olfman, Sein and Bostrom [20]; Olfman [21]).

The second reason that helps explain why there has not been a wider interest in end-user training research is that there has not been a guiding framework for this research. In order to change this situation, Bostrom, Olfman and Sein [2] proposed a framework for understanding the software training/learning process, and identified research issues within this framework. These authors have been carrying out research to validate and enhance the framework. This paper presents the results of one of the research studies. The study compares two software training methods for a popular end-user modelling language, IFPS (Interactive Financial Planning System -- Execucom [8]), taking into account some important individual differences of the trainees.

The next section of this paper briefly outlines the research framework and describes previous related research. The following section presents the specific research questions. Then, the research method is described in terms of subjects, procedures, dependent measures, and data analysis. This is followed by a presentation and discussion of study results. The final section of the paper provides conclusions and implications for practitioners and researchers.

2. PRIOR RESEARCH

The relevant task for trainers is to design effective training environments. These environments consist of the physical characteristics of the training setting, the methods to be used in conveying the material to be learned. The focus of the research reported in this paper is on the design of training methods for a specific training setting, the seminar/classroom.

Figure 1 (Sein, Olfman and Bostrom [25]) summarizes the determinants of end-user behavior. The end-user is a person who is engaged in goal-oriented tasks such as an investment analyst who must choose between alternative investments. In order to use software such as IFPS to support such a task, the user must have the motivation to do so. If the person chooses to use the software, the goal is for him/her to use it as effortlessly as possible. There are certain constraints in this process imposed by what the user understands about the software, by the interface between the user and the software, and by other individual differences such as previous experience with computer software.

Figure 2 (Bostrom et al. [2]) shows the relationship between training and use of the software (problem solving) on the job. In Step 1, Training, the inputs to the training process are the training environment, the system/software to be learned, and the characteristics of the trainee. In the present, if a person has access to a particular software package, then the user-system interface for that package cannot be altered. In addition, this person has a specific
set of characteristics (i.e., cognitive traits, motivational traits, referent experience with computers, and task domain knowledge) that will be utilized in the learning process. Effective training via the training environment must be used as the agent to bring about changes in trainees' levels of motivation to use and understanding of the target software. These outcomes will be reflected in Step 2, Problem Solving, and in subsequent task related uses of the software.

By holding the system/software constant, and comparing training methods, the framework predicts that training outcomes will depend on one or more of the following factors: (1) the abilities of the trainees, (2) the method, and/or (3) the interaction between method and abilities. A recent study by Olfman et al. [20] concluded that basic cognitive abilities were not enough to predict understanding outcomes of end-user software training. The authors used multiple regression to relate 7 basic cognitive abilities (e.g., field independence) that had been shown to be important components of learning, or that were related to skills required for software learning. They found that only a subset of abilities produced significant Beta coefficients, and that no equation explained more than 8% of the variance in the test scores.

While abilities are one source of predicting outcomes for learning, a second source is the training method. The Olfman et al. [20] study looked at the effectiveness of different training methods in terms of the utilization of conceptual models (Moran [19]). It was found that trainees who received a conceptual model as an advance organizer performed better on test questions related to knowledge of the target software architecture. Architecture is one of three components of understanding that was tested. It relates to knowledge about the relationship between subsystems. The other two components, syntax and function understanding showed no difference. They relate to knowledge about command language structure and use, respectively. There were no interaction effects between basic cognitive abilities and the type of conceptual model received.

The use of conceptual models is one way of enhancing understanding outcomes through training method design. It is generally aimed at providing an accurate initial understanding of the target software. It can be embedded within a classroom/seminar session that includes an introduction, presentations, exercises, and documentation.

The trainees in the Olfman et al. [20] study who received no conceptual model were given one of two different methods of training that varied the types of presentations, exercises, and documentation. One method, construct-based training was geared to providing conventional training by using generic examples and by focusing on software syntax and functions. The other method, applications-based training, required students to
bring their own problems to the training sessions, and focused presentations on problem-solving.

The methods were designed to differentiate the level of "personal relevance." Personal relevance refers to the meaningfulness that a particular learning situation has for a trainee. This concept is especially important in the learning process and in the formation and change of attitudes (Ross, McCormick and Krisak [23]; Petty and Cacioppo [22]). The underlying premise is that individuals build their knowledge structures and attitudes on existing scaffoldings. These scaffoldings are highly individualistic, because they are developed through experience. A learning environment that requires trainees to solve their own problems during training can be expected to enhance certain training outcomes.

Training that emphasizes the solution of an individual's task related problem has been touted and refuted in the literature. Some researchers claim that the "narrow" focus of such training is negative because it does not provide a wide enough knowledge base (understanding) for trainees (Bikson and Guitek [1]; Charney and Reder [5]). Some practitioners indicate that they have successfully utilized the "narrow" approach (Gruhn and Hohl [12]; Karten [16]). It is possible that their success can be traced to the motivational effectiveness of using personally relevant software training.

3. HYPOTHESES

One of the study goals was to compare applications-based and construct-based training in terms of understanding outcomes. Construct-based training showed enhanced understanding of syntax and function because of the nature of this type of training. However, it was expected that there would be no difference in terms of understanding of architecture. This is because the Offman [20] study showed that it was a conceptual model that facilitated understanding of architecture. The trainees in the current study did not receive a conceptual model during training. It was expected that, overall, construct-based trainees would demonstrate better understanding than applications-based trainees.

A second goal of the study was to determine if certain individual differences that have been identified as predictors of software training outcomes would moderate the differences in methods. The most consistent individual difference that has been identified in previous software facilitated understanding of architecture. The trainees in during training. It was expected that, overall, construct-based trainees in terms of their task domain knowledge, and for construct-based trainees in terms of their existing knowledge about computer software. It was expected that applications-based training would provide assistance for those with low fluid ability because they were being helped through the use of their own problems.

4. METHOD

4.1. Subjects

Subjects were 64 students enrolled in an introductory half-semester MBA computer tools course in a large midwestern university. In lieu of traditional classroom instruction, each student attended a one-day training session for the IFPS modelling language. The software was implemented on a VAX 11/780 minicomputer. Almost all students had taken at least one computer course, but none had completed a course in IFPS or a similar modelling language. Data is reported for 51 subjects who completed all of the independent and dependent measures in the study.

4.2. Procedure

During an introductory full class session, students completed a questionnaire about their previous computer experience, and were given the (timed) cognitive tests. The questionnaire asked for information concerning previous courses, training, and computer usage. One item, number of hours of current computer use, was used as the measure of experience. Students with more than 6 hours of current use were classified as having "HIGH" experience, while those with 6 or less hours were classified as "LOW".

The cognitive tests were taken from the ETS battery of factor-referenced cognitive tests (Ekstrom, French and Haiman [7]). This battery has been widely used by researchers in the field of educational psychology to predict learning outcomes and interactions with instructional method (for a review, see Federico [9]). The two factors of interest for this study are 'associational fluency' (FA) and 'induction' (I).

Associational fluency is the ability to produce rapidly words which share a given area of meaning or some other common semantic property. It is considered a component of crystallized intelligence (Cattell [4]). Associational fluency was measured with the Controlled Associations test (Ekstrom et al. [7]). Induction identifies the kinds of reasoning abilities involved in forming and trying out hypotheses that will fill a set of data. It is a component of fluid intelligence (Cattell [4]), and was measured by the Letter Sets test (Ekstrom et al. [7]). In both cases, the subjects were classified as having "HIGH" or "LOW" intelligence, with those scoring at or above the mean being classified as "HIGH".

Training was given during a one-day (7 total hours) session that included individual hand-on use of a terminal that was linked to the VAX minicomputer. The format of each type of session is described below. Each session had approximately eight trainees. A total of eight sessions were conducted, four of each type.
Construct-based training provided an introductory discussion on modelling that included an overview presentation of various features and commands of the software. The rest of the session alternated between trainer presentations and hands-on use. The trainer explained how to create constructs and use commands, and used generic examples to do so. The exercises were specific problems that were based on a generic income statement/budgeting problem. Trainees were not guided through solutions to the exercises. The documentation consisted of a manual that specified how to perform various commands, as well as the functions of the commands. The manual was organized around the training session topics.

Applications-based training provided an identical introduction to modelling used in construct-based training, however the demonstration was geared to showing how to solve a problem using the target software. The rest of the session alternated between trainer presentations and hands-on use. The presentation material was aimed at guiding problem solving by working through the solution to the income statement problem used as the exercises in construct-based training. As mentioned above, the construct-based training presentations did not guide problem solving. The exercises for applications-based training were specific problems that the trainees had been directed to bring to the training session. Trainees were given a manual that provided a set of dialogues that were used in the presentation of the income statement/budgeting problem.

Students had two requirements for their course grade, other than attendance at the training session. The first was to complete a take-home assignment using IFPS. The construct-based group received a common question concerning forecasting and budgeting. The applications-based group were told to enhance their own problems. The problem difficulty for applications-based trainees' problems was generally equivalent to that of the construct-based problem. This equivalence was accomplished by the instructor who limited the scope of the applications-based problems during the training session.

The second requirement was a 20 item, 75 minute open-book, open-notes multiple choice exam that was completed at the end of the course.

4.3 Dependent Measures

The exam was used to measure students' comprehension of IFPS. The items in the test represented three dimensions of IFPS knowledge:

(a) Architecture - the relationships between different subsystems of IFPS (6 items)
(b) Function - how IFPS commands are used (8 items)
(c) Syntax - the structure of IFPS commands (7 items)

One item was applicable to both the function and syntax dimensions. The architecture and function dimensions reflect users' understanding of the semantics of the IFPS language. The syntax dimension is analogous to the concept of technical details (Mayer and Bromage [18]).

The test items were chosen in order to sample material covered during the training sessions. The questions were also designed to examine the students' knowledge of the key concepts associated with understanding IFPS.

4.4 Analysis of Data

The ANOVA and CROSSTABS procedures from SPSS® Release 2.1+ (SPSS [20]) were used to analyze all data. A 0.10 level of significance was used to evaluate the statistical tests. It is recognized that the study was exploratory in nature since the training methods were being used for the first time as experimental treatments. The purpose in choosing the significance level is to help highlight the most interesting relationships (or lack of relationships) uncovered by the data.

5. RESULTS AND DISCUSSION

Two sets of data analysis were performed. First, three-way ANOVAs using method, crystallized intelligence, and fluid intelligence were run for each of the four dependent measures (architecture, function, syntax, and total scores). This analysis was designed to compare the

<table>
<thead>
<tr>
<th>Table 1: Cell Means for Each Dependent Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training Method</td>
</tr>
<tr>
<td>Construct-Based (N=30)</td>
</tr>
<tr>
<td>Applications-Based (21)</td>
</tr>
<tr>
<td>Fluid Intelligence</td>
</tr>
<tr>
<td>Low (23)</td>
</tr>
<tr>
<td>High (28)</td>
</tr>
<tr>
<td>Crystallized Intelligence</td>
</tr>
<tr>
<td>Low (25)</td>
</tr>
<tr>
<td>High (26)</td>
</tr>
<tr>
<td>Previous Experience</td>
</tr>
<tr>
<td>Low (26)</td>
</tr>
<tr>
<td>High (24)</td>
</tr>
<tr>
<td>Overall (51)</td>
</tr>
</tbody>
</table>

81
training methods, taking into consideration the level of cognitive ability of the subjects. Second, two-way ANOVAs using method and previous computer experience were run for each of the four dependent measures. This analysis was used to compare training methods, taking into consideration the effect of previous experience.

Table 1 shows the cell means for each of the independent variables (and levels) for each dependent measure. Table 2 shows the results of the three-way ANOVAs, and the two-way ANOVAs in terms of p-values. The results are discussed below.

### 5.1. Comparison of Training Methods

Method showed a significant main effect in each of the two function score ANOVAs, where construct-based trainees scored higher than applications-based trainees. The questions concerning the function of IFPS were designed to test the students' ability to interpret how different language constructs should be used. Because construct-based training aimed to teach and exercise how to recognize different uses of IFPS statements and keywords, the outcome was as expected. An analysis of each of the 8 function questions showed that construct-based trainees had a much stronger understanding of semantics that were specifically exercised in construct-based training. Applications-based trainees may or may not have encountered these semantics in their own problems, so their overall function understanding was lower, as expected.

Figure 3 contains an example of the function-related question that showed the most difference between methods. The question was designed to test whether students understood that using the PREVIOUS keyword in reference to its own variable definition would cause a zero value to be placed in the first column of the model. Construct-based training sought to demonstrate this functional situation. In applications-based training, a trainee would not specifically have encountered this usage of the PREVIOUS keyword.

Architecture and syntax understanding scores were not different between methods. It was expected that architecture scores would not be different between methods, since this type of knowledge is language specific, and can be gained by interaction with the software. Both groups had the same opportunity to use the system during and after training. Average self-reported usage on the post-training take-home assignment was approximately 14 hours. And, although applications-based trainees reported

---

**Table 2: ANOVA p-values by Dependent Measure**

<table>
<thead>
<tr>
<th>Source of Variation</th>
<th>Architecture Score (Maximum=6)</th>
<th>Function Score (Maximum=7)</th>
<th>Syntax Score (Maximum=4)</th>
<th>Overall Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three-way ANOVA</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>METHOD - Training method (2 levels)</td>
<td>0.837</td>
<td>0.070+</td>
<td>0.765</td>
<td>0.471</td>
</tr>
<tr>
<td>GF - Fluid intelligence (2 levels)</td>
<td>0.731</td>
<td>0.884</td>
<td>0.189*</td>
<td>0.421</td>
</tr>
<tr>
<td>GC - Crystallized intelligence (2 levels)</td>
<td>0.243</td>
<td>0.887</td>
<td>0.047*</td>
<td>0.139*</td>
</tr>
<tr>
<td>METHOD x GF</td>
<td>0.851</td>
<td>0.932</td>
<td>0.118*</td>
<td>0.449</td>
</tr>
<tr>
<td>METHOD x GC</td>
<td>0.393</td>
<td>0.500</td>
<td>0.169*</td>
<td>0.885</td>
</tr>
<tr>
<td>GF x GC</td>
<td>0.836</td>
<td>0.888</td>
<td>0.833</td>
<td>0.795</td>
</tr>
<tr>
<td>METHOD x GF x GC</td>
<td>0.063</td>
<td>0.121*</td>
<td>0.980</td>
<td>0.580</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Two-way ANOVA</th>
<th>Architectural Score (Maximum=6)</th>
<th>Function Score (Maximum=7)</th>
<th>Syntax Score (Maximum=4)</th>
<th>Overall Test Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>0.896</td>
<td>0.054+</td>
<td>0.735</td>
<td>0.430</td>
</tr>
<tr>
<td>PCE - Previous computer experience (2 levels)</td>
<td>0.689</td>
<td>0.714</td>
<td>0.082+</td>
<td>0.424</td>
</tr>
<tr>
<td>METHOD x PCE</td>
<td>0.930</td>
<td>0.165*</td>
<td>0.303</td>
<td>0.455</td>
</tr>
</tbody>
</table>

---

**Figure 3: Example of a Function Test Item**

Consider the following segment of an IFPS model:

```plaintext
10 COLUMNS 1-4
15 BOARD = 500
20 FOOD = PREVIOUS*1.2 FOR 2, PREVIOUS
```

Which of the following is true about this model segment?

a. Line 20 is incorrect and could be replaced by `FOOD = PREVIOUS, PREVIOUS*1.2`

b. Line 20 is incorrect and could be replaced by `FOOD = PREVIOUS FOR 2, PREVIOUS*1.2`

c. Line 20 is incorrect and could be replace by `FOOD = PREVIOUS BOARD FOR 2, PREVIOUS`

d. All of the above are true

e. Line 20 is correct.

[f] Item c is the correct answer.
Table 3: Cell Means and [Cell Sizes] for Near Significant Two-Way Interactions for Syntax Scores

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Construct-based</th>
<th>Applications-based</th>
<th>Fluid Intelligence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.73</td>
<td>2.80</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[15]</td>
<td>[15]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.25</td>
<td>3.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[8]</td>
<td>[13]</td>
<td></td>
</tr>
</tbody>
</table>

It was expected that there would be a significant fluid intelligence effect. While this cognitive ability did not show significant differences, the general tendency was that those with "HIGH" fluid intelligence had higher scores than those with "LOW" fluid intelligence. This lack of significance may have been due to the allocation method used to assign subjects to the "LOW" and "HIGH" groups. It may also result from the fact that the tests used in this study measure only one dimension of fluid intelligence. Measures across a number of dimensions may have led to further polarization of the trainees into the "LOW" and "HIGH" categories.

5.2. Role of Individual Differences

The analysis showed main effects for crystallized intelligence (in the three-way ANOVA) and previous computer experience (in the two-way ANOVA) for syntax score. In terms of crystallized intelligence, it was the "LOW" group that scored better than the "HIGH" group. This pattern approached significance for total score. A similar result was found by Olfman et al. [20]. In that study, the authors suggested that one of the reasons for this result could have been the allocation of subjects to groups, and again this may have been true in the present analysis. However, another explanation is that a strong existing organization of specific knowledge may inhibit the learning of new and somewhat different knowledge. The "LOW" group consistently scored higher than the "HIGH" group on all function questions, although no significant differences were found between groups on any one question.

In terms of previous computer experience, the "HIGH" group consistently scored better than the "LOW" group, as expected. A chi-square test was run to test the possibility that the somewhat unexpected intelligence pattern was related to the fact that this group was made up of subjects with mostly "HIGH" previous experience. No significant differences were found (p=.579).

Table 4: Cell Means and [Cell Sizes] of Near Significant Two-Way Interaction for Function Score

<table>
<thead>
<tr>
<th>Training Method</th>
<th>Construct-based</th>
<th>Applications-based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td></td>
<td>4.06</td>
<td>3.50</td>
</tr>
<tr>
<td></td>
<td>[16]</td>
<td>[14]</td>
</tr>
<tr>
<td></td>
<td>2.80</td>
<td>3.30</td>
</tr>
<tr>
<td></td>
<td>[10]</td>
<td>[10]</td>
</tr>
</tbody>
</table>

Another near significant interaction appeared in terms of function score (see Table 4). Those with "LOW"
previous computer experience scored much higher when they took construct-based training compared to applications-based training. In fact, for those who received construct-based training, the "LOW" previous experience group scored higher than the "HIGH" group on function scores. The focus of construct-based training on specific items that also appeared in the test appears to have facilitated this level of performance.

6. IMPLICATIONS AND CONCLUSION

It is clear that individual differences do play a role in learning. The fact that those with "LOW" crystallized intelligence scored higher than those with "HIGH" crystallized intelligence indicates that trainers should consider some methods to un-freeze those with "HIGH" crystallized intelligence prior to and/or during the training session. Applications-based training may provide some help in this regard, at least for learning of new syntax. On the other hand, those with "LOW" fluid intelligence may be better served in a construct-based training session.

Although construct-based trainees scored higher than applications-based trainees on function related questions, the differences were in the opposite direction for syntax and architecture scores. The function score differences may have been, in part, an artifact of the test because the test focused on generic questions that were more familiar to the construct-based group. Therefore, a strong recommendation concerning the use of either type of training is not warranted here. It should be recognized that understanding outcomes, which were tested in this study, are only one of the outcomes of training. Motivation to use the software after training is another important outcome that may be enhanced with applications-based training.

This study further emphasizes that learning to use end-user software is not easy. The average test scores reported here were less than 50%. The test was an open-book exam given to graduate students who had an average of 20 hours of hands-on use with the software. Moreover, only one type of understanding was examined. An additional measure of understanding, in terms of ability to solve a problem, is also an important outcome. In fact, ability to solve problems could be a key factor in developing a trainee's understanding of and motivation to use the software on the job.

Future research in this domain needs to probe the issues discussed above. We have conducted additional studies to examine some of these as well as other questions related to end-user software training (Ofman [21]; Sein [26]). Results of such studies will provide new insights and directions for research and practitioners. We encourage others to explore our framework and studies, and to take on research in this critical area of information systems.

REFERENCES


