On end-user computing productivity Results of controlled experiments

By: Prashant Palvia

Palvia, P. "On End User Productivity: Results of Controlled Experiments," Information & Management, Vol. 21, 1991, pp. 217-224.

Abstract:
Two important classes of end-users are: command level users, and end-user programmers. For each of these classes, controlled experiments have been conducted to measure their productivity. For command level users working with databases, our results indicate that they perform better when the logical data model used is consistent with their own views of reality. For end-user programmers writing programs in high-level procedural programming language, the quality of programs is generally low and no program development method clearly stands out in terms of quality; though writing programs directly without any preplanning is expeditious.
Keywords: End-users, Productivity, Database, Programming, Information systems, Data models, System quality, System evaluation, End-user computing, Programming languages

Article:

Introduction

End-user computing has undergone explosive growth in the past few years [1,3,4,7,23,24]. In their seminal article, Rockart and Flannery [24] discussed several fundamental issues concerned with end-user computing. They categorized end users in six categories. Two important ones (perhaps constituting the majority of end-users) are command level users and end-user programmers. Others [12,22] have discussed the role of end-users as developers of IS (information system) applications and database applications. Pitfalls and risks associated with end-user computing have been discussed in [2,11]. Strategic implications of end-user computing have been envisioned in [14].

This article attempts to evaluate, objectively, the effectiveness of end-users as programmers and command level users. In the command level mode, end-users need to access data on their own terms, and perform simple inquiries often with simple calculations [24]. In the programming mode, end-users develop their own IS applications using a high-level procedural language, a fourth generation language (4GL), or a language which is part of a software package. While different "levels" of end-users exist, in this article we are considering end-users who are "non-proficient" and relatively unexposed to data-processing/ programming. Many end-users meet this characterization.

Two sets of controlled experiments were conducted, one for each type of end-user mode. The method of controlled experiments is generally considered a good research tool, yielding excellent internal validity. Several research articles based on experimental methodology have been written in the IS literature [16,17,26,27]. For easy reference, we will refer to the two experiments as the "database experiment" and the "programming experiment".

Research Objectives

The overall goal of the two experiments is to measure the various elements of productivity of both command level users, and end-user programmers. A useful definition of the productivity of a process is [5]:

$$\text{Productivity} = \frac{\text{Outputs provided by the process}}{\text{Inputs consumed by the process}}$$

The elements of productivity addressed in this article include: outputs, inputs, and productivity itself. In both experiments, input is readily defined as the time taken to complete the given task. Surrogate measures have been used for measuring output. We describe the motivation, objectives and surrogate measures for each experiment below.

The Database Experiment

Our focus in this study is to examine the effectiveness of non-proficient end-users in working with databases. In terms of previous work, Juhn and Naumann [16] examined the effectiveness of different schema representations based on different data models; however, end-users seldom work with schemas but work directly with actual data. Hoffer [15] looked at the relationships between the ways subjects perceived data architecture, and cognitive style and DP background, and had either insignificant or mixed results. Again, end-users generally do not have the data architecture options elicited in Hoffer's work and usually have to conform to what is available. Finally, a recent study by Boehm-Davis et al [6] examined the effects of three database formats: spatial, tabular, and verbal on the retrieval performance of users. While, this study indicated the superiority of each format under unique user requirements, currently commercial databases management systems (DBMS) typically do not provide spatial or verbal formats. Most commercial DBMS are "tabular" in nature and are based on prevalent logical data models.

To a command level end-user, a database may be expressed using one of several logical data models. The three established data models that exist today are the hierarchical model [19,21], network model [8,25], and the relational model [9,10,19,21]. In recent years, the relational model has gained wider acceptance because of several user-oriented features [10]. We included a fourth model in our study, the object-oriented model, where the logical representation of data matches the user's perception of reality. The object-oriented model is currently being emphasized in the IS literature, and has even appeared in some text books (e.g. [19]).

Command level users primarily interface with a database and retrieve information from it. Given a database, the output of the users can indirectly be measured by their understanding of the meaning of the various data elements and data relationships present in the database. This " understanding" is, therefore, our surrogate measure of output.

The following productivity-related objectives are addressed in this experiment:

1. Validate the end-user understanding of the meaning of the database for the database expressed in each of the logical data models.

2. Compare the logical models on the basis of end-user understanding.

3. Compare the efficiency of the end-user (as measured by the time to complete a given task) in understanding the database, using the different logical models.

4. Compare the productivity of the end-user for the different logical models.

The Programming Experiment

The quality of computer programs developed by end-users is generally suspect, and many examples exist of the consequences [11]. This may not be a serious problem when end-users use well-tested software packages (e.g., spreadsheets, database systems, graphics, etc.). But, when the end-users develop their own computer programs using a high-level procedural language (without the benefit of formal training in logic development and programming), the quality and accuracy of these programs may be compromised. As such, the goal of this experiment is to evaluate the productivity of end-users when they write programs in a high level language. Previous studies have generally focused on computer program development methods in the context of programmers or more experienced users [13,17,26].

The surrogate measure of the output aspect of productivity used in this experiment is the quality of the program. Quality is broken into three components: accuracy of the program, degree of structure in the program [18], and efficiency of the program. Efficiency of the program is measured in several ways, e.g., the length of code, the memory requirements, and the CPU time to execute. Since end-user programs are generally short and ad hoc, the concern for the efficiency of programs is not overriding.

Table 1

Subject Profile for the Database Experiment
Total number of subjects: 149

SUBJECT'S YEAR IN SCHOOL		GENDER	
Freshman:	4.0%	Male:	52.3%
Sophomore:	20.1%	Female:	47.7%
Junior:	57.7%		
Senior:	16.2%		
Graduate student:	2.0%		

JOB CLASS	
Management:	20.0%
Clerical:	39.3%
Sales:	9.3%
Analyst:	0.7%
Labor:	8.6%
Other:	22.1%

Mean GPA = 2.80, standard deviation = 0.47
Mean Number of Computer courses = 1.19, std. dev. = 0.82
Median total experience = 48 months, std. dev. = 29.3
Median computer experience = 3.5 months, std. dev. = 16.2

Subject Profile for the Programming Experiment
Total number of subjects: 83

SUBJECT'S YEAR IN SCHOOL		GENDER	
Sophomore:	25.6%	GENDER	
Junior:	46.4%	Male:	57.8%
Senior:	14.6%	Female:	42.2%
Graduate student:	13.4%		

JOB CLASS		FUNCTIONAL AREA	
Management:	18.7%	Accounting:	13.5%
Clerical:	30.7%	Finance:	23.2%
Sales:	17.3%	Marketing:	13.4%
Analyst:	2.7%	MIS:	4.9%
Labor:	17.3%	Management:	24.4%
Data Proc.	4.0%	Real Estate:	1.2%
Other:	9.3%	Other:	19.4%

Mean GPA = 2.65, standard deviation = 0.79
Mean Number of Computer courses = 1.96, std. dev. = 2.02
Median total experience = 48 months, std. dev. = 52.77
Median computer experience = 12 months, std. dev. = 24.55

There is no single dominant method of pro-gram development. There are several methods in use, and several that are recommended (both at various levels of complexity). To get a realistic assessment of program development by end-users, the following four are considered:

1. Develop the logic of the underlying task using a flow diagram and then write the computer pro-gram.

2. Develop the logic of the underlying task using pseudocode (also called "structured description") and then write the computer program.

3. Develop the logic of the underlying task using narrative description and then write the computer program.

4. Do not formally develop the logic of the program, instead directly write the computer program.

Given these methods, the following productivity-related objectives are addressed in this experiment:

1. Measure the quality of end-user written computer programs written in procedural language. Quality is further broken into three components: accuracy of the program, degree of structure of the program, and efficiency of the program.

2. Compare the four program development methods on quality components.

3. Compare the four program development methods on end-user efficiency (as measured by the time taken to complete a given programming task).

4. Compare the productivity of the end-user with the four program development methods.

Research Methodology

As stated earlier, experiments were the basis for data collection. Controlled experiments were conducted on subjects who were students in an introductory computers/MIS class in the business school of a major American university.

We have reasons to believe that the subjects reflected characteristics of most end-users. Just as end-users are not experts in data processing or programming, these subjects were also not experts as this was their first course introducing them to computers, MIS, and programming. All end-users are somewhat computer-literate and these subjects were by the time the experiment was conducted. It was conducted in the last week of the semester; by then, they had been introduced to basic concepts in computers, MIS, and some programming. Most of the subjects were either currently holding full-time jobs or had prior job experience. The demographic profiles of the subjects in each of the two experiments are shown in Table 1.

Now, we describe the specific and unique features of the methodology used for each experiment.

The Database Experiment

An organizational database problem was developed. It consisted of three entity classes: Department, Employee, and Project. The relationship between Department and Employee is one to many, and the relationship between Employee and Project is many to many (see the corresponding data structure diagram in *Figure 1).* Based on this problem, databases based on several logical models were developed containing actual data values. For each logical model, data files were created and displayed to the subjects on paper.

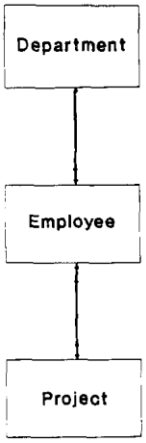

Fig. 1. Data Structure Diagram for an Organizational Database.

Each subject was given the same database rep-resented in one of the logical data models. In addition, they were given an examination on the database. The examination consisted of thirty-two questions, twenty-five of which were on the understanding of the database and five asking them to make modifications to the database. There was one question asking them to express their confidence in the answers and one requiring them to report the time taken to complete the exam. In addition, they had to complete a brief questionnaire seeking demographic data. Participation in the experiment was mandatory, and each subject received partial credit towards the final grade.

Each exam was scored. For the "understanding" type questions, the subject received either zero or one point; however, partial scores were given for the "modification" type questions. As stated earlier, this article reports on the understanding of the database and the time to complete the given task.

The Programming Experiment

Each subject was given the same "moderate" amount of training in writing simple programs. Instruction and practice were given in class in the development of flow diagrams, pseudocode, and concepts of structured programming. This training lasted about three weeks of class lectures, and was considered adequate for writing small programs. In addition, all subjects also wrote three computer programs in BASIC programming language. BASIC was used as the programming language in the experiments, because it is the major language available on personal computers and predominantly used by end-users.

For the experiment itself, a relatively simple problem task was developed. It required the three basic structures of programming (i.e., sequence, selection and iteration [181). This problem was given to all subjects, who were divided into four groups of about equal size and were asked to write the computer program using one of the four techniques.

Each subject also completed a short questionnaire about demographic data and reported the time to complete the program. Again, participation was mandatory and students received partial credit towards the final grade.
Each response was evaluated by two graduate assistants. Only the final product of the program development process, i.e., the actual hand-written program, was evaluated. Each graduate assistant graded each program on its degree of structure (on a 1 to 5 scale; 1 being no or low structure and 5 being very high structure), correctness of the program (again on a 1 to 5 scale), lines of code (not counting comment lines), and finally a grade for the overall quality of the program (on a 0 to 10 scale). Since grading involves some subjectivity, we used two graders for each program. If the grades assigned by the two graders were close, then we simply took their average; otherwise, the author arbitrated, regraded the program, and then assigned a grade.

Finally, several statistical tests were conducted for both experiments to analyze the data and produce the results.

Results

The Database Experiment

As a caveat, it should be realized that the subject's performance with each data model is influenced to a degree by the type of questions asked. As reported by Boehm-Davis et al. [6], users perform better when the form of the data-base matches the type of questions. However, in our study, we have included a variety of question types that are typical of business data processing, and the purpose is to evaluate the overall performance of the end-user with each model. Also, our results are less sensitive to this effect, as our data models are not as radically different as in [6] (they considered spatial, tabular and verbal models, while our models may be considered variations of the " tabular" model).

1. End-User Understanding of the Meaning of the Database:

There were a total of 149 usable responses from the subjects: 28 used the relational model, 18 used the hierarchical model, 29 used the network model, 20 used the object-oriented model, and the remaining used other models. Since, the database problem was relatively small, most subjects did a good job. On understanding based on retrieval of information, the average score was 23.95 out of a maximum of 25 and the standard

deviation was 1.24. There were five somewhat complex query questions; on those the scores were not as high. The time to complete the task averaged 29 minutes.

2. Comparison of the Four Logical Data Models on Understanding:

The average scores on all 25 questions, are reported below.

	Hierarchical	*Network*	*Relational*	*Object-Oriented*
Average:	23.439	24.287	23.605	24.037
Std. Dev.	1.750	0.850	1.103	0.841

Using an ANOVA test for differences of means, these results are only marginally significant (p value of 0.168). On specific complex questions, the p value ranges between 0.026 and 0.06. Interpreting the results, the network and object-oriented data models yield a higher understanding of the data than the hierarchical and relational models. The underlying reason may be that the object-oriented model and the network model represent the data closer to the user's way of viewing his/her data needs than do the relational model and the hierarchical model. For example, the user is not restricted to only 1 : M relationships or non-repeating group data items.

3. Comparison of the Four Logical Data Models on Efficiency:

The average time (in minutes) required to complete the task using each of the four models is shown below.

	Hierarchical	*Network*	*Relational*	*Object-Oriented*
Average:	32.7	28.1	32.4	26.8
Std. Dev.	8.1	6.7	5.3	7.1

Again using ANOVA, these results are highly significant (p value of 0.001). As before, network and object-oriented models take significantly less time for task accomplishment than hierarchical and relational models. This comparative ease of task accomplishment may again be due to the closeness of the network and object-oriented models to the user's way of thinking about this problem.

4. Comparison of the Four Logical Data Models on Productivity..

Dividing the understanding score by the time taken to complete the task gives us a measure of productivity. The productivity measures for the four logical models are:

	Hierarchical	*Network*	*Relational*	*Object-Oriented*
Average:	0.782	0.908	0.746	0.959
Std. Dev.	0.297	0.203	0.115	0.261

Again the statistical tests show that these means are significantly different (p value of 0.001). The object-oriented model is the most productive, closely followed by the network model. The hierarchical and relational models are the least productive for this problem.

The Programming Experiment

1. Quality of End-User Written Procedural Pro-grams:

Three measures were captured which reflect the quality of the end-user programs: structure, correctness (accuracy) and overall quality. The total number of subjects for this experiment was 83. On a 1 to 5 scale (1 being low and 5 being high), the average structure of the end-user written programs was 2.17, with a standard deviation of 1.09. The average correctness score, again on a 1 to 5 scale, was 2.27 with a standard deviation of 1.07. The overall quality score, on a 0 to 10 grading scale, was 4.13 with a standard deviation of 2.6. These results suggest that the quality of programs, writ-ten in procedural language by end-users, is relatively low. This should not be a totally surprising result; in fact it supports the documented experiences of some researchers in the field [11].

2. Comparison of the Four Methods on the Quality of Programs:

As previously stated, method 1 (M1) is to develop a program after developing a flow diagram, method 2 (M2) is to develop a program after developing a pseudocode, method 3 (M3) is to develop a program after writing a narrative description, and method 4 (M4) is to directly write a program. Our initial hypotheses on quality were:

a. End users writing a program directly without the use of a formal method in logic development will have the poorest quality program.

b. Programs written with flow charts or pseudo-code for logic development will be of the highest quality.

c. Programs written with narrative description for logic development will be of higher quality than those written directly.

Table 2

	M1	M2	M3	M4
Number of Subjects	25	16	15	27
Average Overall Quality Grade	3.98	4.75	3.73	4.11
Average Structure Score	2.22	2.06	2.20	2.17
Average Correctness Score	2.38	2.25	2.20	2.20

The results of the experiments are shown in Table 2. For each method, results are included on the number of subjects, the average structure score, the average correctness score, and the average overall quality score. The above results do not support any of the above hypotheses, based on ANOVA tests for equality of means. The p values from the ANOVA tests are 0.726, 0.975, 0.935 respectively for overall quality, structure and correctness.

Some mild differences exist between the methods (though not statistically supported). For example, method M2 (i.e. using pseudocode to develop programs) yields the highest overall quality programs. Contrary to expectations, method M4 (writing programs directly) does not, in most cases, produce programs any worse in quality than programs written with formal logic development. Also, there is a slight indication that flow diagrams yield better structured programs; and both flow diagrams and pseudocodes generate slightly more correct programs. Note that these results are to be interpreted in the proper context, i.e., inexperienced end-users writing small programs.

3. Comparison of the Four Methods on End-User Efficiency:

Our hypotheses regarding the efficiency of the methods were:
a. Of the four methods, programs written with flow charts for logic development will take the most time to complete.

Table 3
Time to Complete the Program

Method M1 (using flow diagram):	18.9 minutes
Method M2 (using pseudocode):	16.8 minutes
Method M3 (using narrative description):	14.8 minutes
Method M4 (directly writing program):	13.9 minutes

b. Programs written directly will take the least time to complete.

Both hypotheses are only mildly supported (an ANOVA on the completion time of the four methods gave a p value of 0.18). The rank ordering in descending order of the four methods according to average completion time for the program is: Ml, M2, M3, and M4. The actual average times are reported in Table 3.

4. Comparison of the Four Methods on End-User Productivity:

The above findings on efficiency take special relevance, primarily because we did not find statistically significant differences between the quality of programs written by the four methods. Dividing the overall quality by task completion time gives us a productivity measure. Statistical tests showed significant differences

between the four methods on this productivity measure. Methods M2 (using pseudocode) and M4 (directly writing programs) showed the highest productivity ratings; comparatively, the ratings of the other two methods were low.

Discussion and Conclusions

While the results of controlled experiments are sound in terms of internal validity, their interpretation and application to other external environments need to be done with caution and care. With this caveat, these experimental results have several implications for end-users, whether they are working as command level users or programmers. There are implications for information system designers and database designers also.

As per our first experiment, command level end-users working with databases perform better when they are working with logical data models consistent with their own views of reality. They are more accurate in their understanding, are more efficient, and are more productive when working with databases based on object-oriented or network data models.

The IS or database designer, when designing a database for the end-user use should find this result very important. The technical merits and drawbacks of the data models have been well documented in the literature; however end-user concerns have largely been ignored. We suggest that irrespective of what the physical and global logical organization of the data is, the designer should strive to provide the local interface (sub-schema) of the database to be consistent with the end-user's perception of reality. On this yardstick, and based on our problem, our experiment suggests that the object-oriented and network models hold greater promise than the hierarchical model and the often-touted relational model.

Several lessons are also learned from our second experiment. When end-users write small programs in a high-level procedural language, then no method for program development stands out. The programs are of about equal low quality irrespective of the program development method. Moreover, writing a program directly is expeditious. While intuitively and logically, it does not seem right to write a program without formal development of the logic, our result suggests the opposite. Perhaps, for small problems, the logic development can be done intuitively and formal logic development only adds to the total time for program development.

On productivity measures for programming, the two methods that stand out are program development directly and development with pseudocode. Although direct program development may appear tempting, we recommend the preparation of pseudocode prior to program development. This is consistent with the structured approaches advocated in the literature.

The finding that the quality of the programs written in high-level procedural language by end-users was low to medium was somewhat expected, but still disturbing. One way to alleviate this is by way of formal computer and programming education. However, this approach may be impractical for many organizations, because of the amount of resources required for training. Besides, it defeats the very concept of end-user computing, i.e., the ability of relatively naive users to compute.

The problem lies in the end-users not being able to specify the procedural details of the computer program. The end-users are functional experts, and it should be easy for them to specify the functional requirements of the computer program. Why not give them languages which work largely on the basis of functional requirements? The answer lies in the fourth generation languages and tools (4GL's) [20], which are non-procedural and declarative means of task accomplishment. They rely on the " what" of the problem rather than the " how". Therefore, we recommend that managers of end-user computing should focus on acquiring the 4GL's and should encourage the end-users to use them.

Finally, end-user computing has become so pervasive that it demands continuous scrutiny and attention. While we have addressed specific issues of two constituencies of end-users (command level end-users, and

programming end-users), more studies based on sound research methods need to be undertaken to address other categories of end-users as well as their unique needs.

References

[1] Alavi, M., Phillips, J.S. and Freedman, S.M. Strategies for Control of End-User Computing: Impact on End Users. Proceedings of the International Conference on Information Systems, San Diego, CA 1986.

[2] Alavi, M. and Weiss, I.R. Managing the Risks associated with End-User Computing. *Journal of Management Information Systems, 11,* No. 3, Winter 1985.

[3] Benjamin, R.I. Information Technology in the 1990s: A Long Range Planning Scenario. *MIS Quarterly.* June 1982.

[4] Benson, D.H. A Field Study of End User Computing: Findings and Issues. *Management Information Systems Quarterly,* December 1983, pp. 35-45.

[5] Boehm, B.W. Improving Software Productivity. *IEEE Computer,* September, 1987, pp. 43-57.

[6] Boehm-Davis, D.A., Holt, R.W., Koll, M., Yastrop, G., and Peters, R. Effects of Different Data Base Formats on Information Retrieval. *Human Factors.* Vol. 31, No. 5, 1989, pp. 579-592.

17) Chaney, P.H., Mann, R.I., and Amoroso, D.L. Organizational Factors and End-User Computing. *Journal of Management Information Systems.* Vol. 3, No. 1, 1986, pp. 65-80.

[8] CODASYL Data Base Task Group Report, 1971, Available from ACM, New York,

[9] Codd, E.F. A Relational Model for Large Shared Data Bases. CACM, June 1970.

[10) Date, C.J. *"An Introduction to Database Systems".* Addison-Wesley, 4th Ed., Vol 1, 1985.

[11] Davis, G.B. "Caution: User-Developed Systems Can Be Hazardous to Your Organization". Proc. of the 17th Annual Hawaii Conference on System Sciences. January, 1982.

[12] Gremillion, L.L. and Pyburn, P. Breaking the Systems Development Bottleneck. *Harvard Business Review.* March-April, 1983.

[13] Guimares, T. A Study of Application Program Development Techniques. *Communications of the ACM,* Vol. 28, No. 5. May, 1985.

[14) Henderson, J.C. and Treacy, M.E. Managing End-User Computing for Competitive Advantage. *Sloan Management Review,* Winter 1986.

[15] Hoffer, J.A. An Empirical Investigation into Individual Differences in Database Models. Proc. 3rd Intl Info Sys Conference, Ann Arbor, Mich., Dec. 1982.

[16) Juhn, S.H. and Naumann, J.D. The Effectiveness of Data Representation Characteristics on User Validation. ICIS, 1985.

[171 Kendall, K.E. and Yoo, S. Software Design with the PB (Pseudo-Box) Diagram: Performance Evaluation Using a Quasi-Experimental Method. Proceedings, 1986, National DSI Conference.

[18] Korsh, J.F. Data Structures, Algorithms, and Programming Style. PWS, 1986.

[19] Kronkie, D.M. and Dolan, J.A. *"Database Processing: Fundamentals, Design, Implementation".* 3rd Ed., SRA, 1988.

[20] Martin, J. *Fourth-Generation Languages,* Volume I, Principles, Prentice-Hall, 1985.

[21] McFadden, F.R. and Hoffer, J.A. *"Database Management".* Benjamin Cummings, 1985.

[22] McLean, E.R. End Users as Application Developers. *Management Information Systems Quarterly.* December 1979, pp. 37-46.

[23] Pyburn, P. J. Managing Personal Computer Use: The Role of Corporate Management Information Systems. *Journal of Management Information Systems.* Vol. 3, No. 3,
Winter 1986-87, pp. 49-70.

[24] Roekart, T.F. and Flannery, L.S. The Management of End-User Computing. *Communications of the ACM,* 26, 10, October 1983.

[25] Tsichritzis, D.C. and Klug, A. The ANSI/X3/SPARC DBMS Framework. Report of DBMS Study Group. In-formation Systems, 3.

[26] Vessey, I. and Weber, R. Structured Tools and Conditional Logic: An Empirical Investigation. *CACM,* Jan. 1986. Vol. 29, No. 1.

[27] Wells, C.E. and Naumann, J.D. An Empirical Comparison of User Understandability: System Flow Charts and Data Flow Diagrams. Proceedings of the 1985 Annual Meeting, DSI, Las Vegas.