1993 IEEE Symposium on Visual Languages, Bergen, Norway, Aug. 24-27, 1993, pp. 344-351.

Is It Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study

Rajeev K. Pandey   Margaret M. Burnett

Department of Computer Science, Oregon State University
Corvallis, Oregon 97331 USA

Abstract

Empirical studies comparing the effectiveness of visual languages versus textual languages have rarely been attempted. Here we describe an experiment conducted on programmers solving vector and matrix manipulation tasks using the visual language Forms/3, the textual language Pascal, and a textual matrix manipulation language with the capabilities of APL. Presented here are our motivation, experimental approach, some of the difficulties experienced in attempting this type of empirical study, and a summary of the experimental results and insights gained.

1: Introduction

A long term goal of research in visual programming, according to a 1991 paper by E. P. Glinert et al., is:

> to develop "general purpose" visual languages which combine textual and graphical elements in so powerful and compelling a manner that tomorrow's programmers would prefer them to purely textual languages for their everyday tasks [7].

In order to validate any claims of visual programming languages replacing textual languages for everyday tasks, "programmer preference" must be established empirically. Very few such empirical studies have been attempted to date. But now that research in visual programming languages is maturing, it is becoming critical that visual programming language researchers begin to substantiate claims that visual languages make programming easier. As a step in that direction, we performed an empirical study to determine whether solutions to vector and matrix manipulation tasks were more easily constructed in a visual or a textual programming language.

We were interested in finding out whether the visual programming language Forms/3 [3, 4] was "better" in this problem domain than textual languages. Thus, the textual languages used in the study were chosen so that we could measure the visual language against (1) a "normal" textual language typical of those used most widely, and (2) a textual language that was especially well-suited for the chosen problem domain, namely matrix manipulation. For these reasons, we chose Pascal and a simplified version of APL (called OSU-APL) as the textual languages against which the visual language Forms/3 would be compared. (OSU-APL is described in section 2.)

Our hypothesis was that the visual programming language Forms/3 is better suited for constructing matrix manipulation programs than the textual languages Pascal and OSU-APL. The results of our study supported this hypothesis. To perform our study, after demonstrating the construction of solutions to example vector and matrix manipulation problems in Forms/3, Pascal, and OSU-APL, we posed similar problems for the subjects to solve in these languages. Of the solutions constructed by the subjects, 73% of the Forms/3 solutions were completely correct, while only 53% of the solutions composed in OSU-APL were correct, and less than 40% of the Pascal-based solutions were correct. This result is particularly interesting in light of the fact that all of our subjects had extensive experience with either Pascal or C.

In the following section, we present related work and briefly describe the languages used in this study. Section 3 describes the experiment, section 4 describes some of the results obtained, and section 5 contains a discussion of observations and insights. Our conclusions are presented in section 6.

2: Background

Visual programming is a relatively new area of research, and there are few accounts in the literature of empirical studies measuring their effectiveness. In this section, we describe the only reports that we have been able to locate of empirical studies related to visual programming.

2.1: Related work

In [12], the learning benefits of an algorithm animation system were measured empirically by comparing 10 students' understanding of an algorithm using the animation system versus that of 10 other students who did not have access to the animation system. Their study reported that the use of animations did not help subjects' understanding as much as had been expected, and led to some insights into how the use of algorithm animation might be changed to improve this result.

In [1], the ease with which a programmer can translate task requirements into a visual language called ChemTrains was measured on six subjects using a technique termed a "programming walkthrough". This method analyzes the mental steps required by a programmer, and was used to evaluate different designs of the same language, i.e. to evaluate a visual programming language without comparing it to a textual language.

A study measuring comprehension of dataflow visual programming languages was conducted on five subjects, and reported textual languages to be superior [8]. For visual languages, both LabView 2™, a commercial product by National Instruments Corp., and a generic visual dataflow representation were used. For the textual languages, two different kinds of pseudo-code were used. However, we do not feel it is possible to accurately measure comprehensibility of a dynamic visual language using only a static representation. This will be discussed further in section 5.

A study of comprehension of a static visual flowchart language versus the textual language Pascal was conducted on 23 novice programmers [6]. This study reported that the flow-chart language was much easier to comprehend. Their measurement techniques were based on reaction time and on number of correct responses. There have also been a number of studies that compared flow-chart representations of programs (not visual languages based upon them) to purely textual representations. Since few visual language researchers today are working on flow-chart languages, this group of work is primarily of historical interest. Most of these studies found that flow charts were better for some types of tasks, and textual representations were better for other types of tasks.

2.2: The languages

Forms/3 [3, 4] is a form-based, declarative visual programming language. In Forms/3, programming is done by defining formulas for cells on forms. A research prototype has been implemented on a Sun color workstation using Lucid Common LISP and the CLX interface to the X Window system. Forms/3 includes strong support for matrix operations, which was the area we chose to measure in our experiment. The approach to matrices was developed by Viehstaedt and Ambler [13], which we have modified slightly. Forms/3 also includes facilities for procedural abstraction, visual data abstraction, polymorphic typing with type inference, files, events, and recursion. These features were not measured or included in the study.

Since Forms/3 is a vehicle for visual language research, its syntax is still evolving. Early attempts at presenting the material in our pilot study produced some resistance by the subjects to the syntax we were using for Forms/3 at that time, causing problems with usability. This feedback caused us to revise the syntax to that shown in this paper. This revised version of the syntax is the one which was used for the experiment. The revision seemed to produce an improvement in usability, as is borne out by the results. The syntax also created positive responses from a number of the subjects, many of whom later commented to the authors that they enjoyed the experiment and the presentation of Forms/3.

Pascal [10] is an imperative, block-structured language with well-developed data-structuring and program structuring facilities. Pascal has held a prominent position as a pedagogic vehicle in introductory computer science courses, and its use in introductory courses at our institution influenced our selection of it for purposes of our experiment. A stronger motivation for including Pascal was that it is widely-used, and represents the "standard" way in which textual programming is done. The students were allowed to use any aspect of Pascal except recursion (since recursion was not presented in Forms/3).

The main data structure of APL [9] is the array, making it especially well-suited for matrix manipulation. In APL there are operators for restructuring arrays into new dimensions and sizes, for combining arrays into new arrays, for deleting parts of arrays, for applying standard operators to an array in element-wise fashion, for applying standard operators in cumulative fashion, etc. APL uses a mathematical notation, employing a keyboard with special characters in order to use mathematical symbols.

However, we did not use standard APL in our experiment. Its dense syntax and right-to-left evaluation order have made it inaccessible and unpopular. To retain its advantages but not its disadvantages, we devised an alternative syntax that uses traditional English-based terms and left-to-right evaluation order. In order to distinguish between our version of APL and standard APL, we denote our version as OSU-APL.

An example of using each of these languages to solve a matrix manipulation problem is given in figure 1. Suppose we wish to solve the problem shown in figure 1a. The solutions in Pascal, OSU-APL and Forms/3 are shown in figures 1b, 1c, and 1d respectively. The solution in standard APL is shown in figure 1e. In all cases, the operations being performed are the summation

```
Write a program that computes the mean of an N-element
vector.  For example,
          given the vector 3 3 4 4 4 5 5
          returns 4
```

Figure 1a. Problem statement.

```
function avg (v: Vector, size: integer)
             : integer;
var i, total: integer;
begin
      total := 0;
      for i := 1 to size do
         total := total + v[i];
      avg := total div size;
end;
```

Figure 1b. Finding the mean in Pascal.

```
avg = RowReduce(+, v) / Dimension(C, v)
```

Figure 1c. Finding the mean in OSU-APL. The Dimension operation returns the number of column elements of the vector v (the "C" means "by Columns"), while the RowReduce operation applies the cumulative operation + over the elements of v.

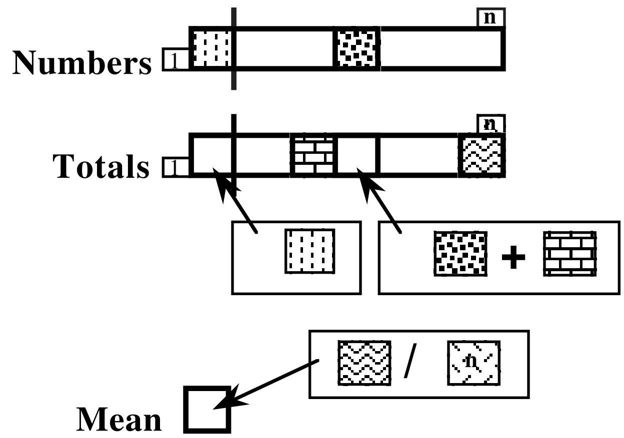

Figure 1d. Finding the mean in Forms/3. The user has created a new vector (Totals) to maintain a running total and divided it into two regions. Each region consists of cells that have like formulas. The formula for any cell in the large region is the previous cell plus the corresponding cell in the original vector (Numbers). The first cell is in a different region, and its formula is a reference to the first cell of Numbers. The sum found in the last cell of Totals, divided by the number of elements (n), yields the mean.

```
average ← (+/ v) ÷ ρ v
```

Figure 1e. Finding the mean in standard APL.

of the elements of the vector followed by a division by the number of elements in the vector.

3: The experiment

An in-class experiment was conducted in which the participants first were given a lecture on the application of Pascal, OSU-APL, and Forms/3, to various matrix manipulation tasks. All subjects were then given a timed test in which they were to compose solutions to two matrix manipulation problems in all three languages.

Brooks [2] gives a number of pitfalls that can occur in such experiments. In experiments of this sort, it is critical to remove any unintended factors that may affect the results. We took a number of measures to avoid those pitfalls. One such pitfall is that studies have shown that significant differences exist among student subjects even at the same scholastic level. To alleviate the possibility of this influencing the results, we chose a within-subject experimental design. This means that all subjects are exposed to all treatments, and attempt to perform all the tasks.

To minimize the possibility that the quality of the explanation of the visual language might affect the outcome, we explained the features needed for the same capabilities in all three languages. If we had explained only the visual language, it would have been possible that a poor explanation would bias the study against it, or an excellent explanation would bias the study in favor of it. Instead, we used a comparative approach, taking care to maintain consistency of the explanations of the three languages.

A common problem in this kind of study is small sample size. It is tenuous to draw conclusions about the usability of anything based on small samples. We were able to use 60 subjects in our experiment, which is somewhat small, but is considerably larger than many usability studies reported in the literature. Still, we would like to repeat the experiment with two or three other large upper-level computer science classes, to see if increasing the size of the study causes the results to change.

One possibility was that the choice of the tasks themselves might influence the outcome. We zealously tried to avoid this, but feared that the tasks were biased in favor of Pascal because they are such common examples in introductory Pascal courses. However, if any such bias toward Pascal was present, it was not enough to make Pascal outperform the visual language.

The entire lecture was presented by one presenter, in one session, to all the participants at once. This ensured that all participants received exactly the same treatment, that the lecturing style would remain constant, and that each single participant's state of mind, general health, etc., would be consistent for the entire experiment.

We presented only constructs to perform exactly the same tasks in each language. Our method to ensure this was to choose example problems, and then solve the problems in each of the three languages. We did not cover any aspects of the languages that were not needed to solve the examples. On the test, all the subjects were given exactly the same 6 test questions, although the order of those questions was varied.

3.1: The lecture

Subjects were given a 40-minute lecture that examined the solutions to various matrix manipulation tasks in the three languages Pascal, OSU-APL, and Forms/3. Example manipulation tasks were first described to the subjects via a slide showing a specific numeric example (as in figure 1a), and then the subjects were shown the solution on three subsequent slides, one slide for each example language (as in figures 1b-d). The example matrix manipulation tasks chosen for the lecture were:

- incrementing each element of a matrix
- creating a vector of k elements, with each element equal to some number n
- counting the number of negative elements in a vector
- computing the sum of a vector
- computing the mean value of the elements of a vector (figure 1)
- rotating the columns of a matrix

During the lecture, we only described the language constructs necessary to adequately explain each example. By selectively explaining only concepts used in each example, we were able to retain the attention of the subjects. Care was taken in choosing the examples to ensure that the subjects possessed all concepts necessary to solve the matrix manipulation tasks posed to them during the evaluation phase. By the end of the lecture, all of the key concepts in each language regarding matrix and vector manipulation tasks had been presented. Due to the amount of time allocated to the lecture, we did not attempt to present any aspects of the languages other than those needed to solve matrix and vector manipulation tasks.

3.2: The subjects

Because the goal of this experiment was to compare programmers, we used upper-level computer science student subjects in this study. All of the students were drawn from the 1993 Winter Term CS 381 (Programming Languages) course. As previously stated, all students were subject to the same treatment. Prior to conducting the experiment, the students took a survey in order for some demographic information to be compiled regarding the subjects' abilities and experiences.

There were 60 student participants. More than 60% of the participants were seniors, and more than 90% of the participants were juniors or seniors. The remainder of the students were graduate students, post-baccalaureate students, and sophomores. Only one participant had prior experience with APL, none had prior experience with Forms/3, all participants had prior experience with Pascal or C (or both), and all had prior experience with Lisp.

3.3: The procedure

Prior to the beginning of the 40-minute lecture, students were given copies of the overhead transparencies to be used so that they could take notes on them, and were allowed to refer to these notes during the test. We felt the active use of the transparency copies for note-taking would improve their learning ability, and also would remove short-term memorization skills as a factor in the results.

Upon completion of the lecture, the students were tested. The test materials consisted of two matrix manipulation problems phrased in the same manner as the examples employed in the lecture. For each problem, the students were required to construct solutions (on paper) in each of the three languages. Thus, all the students composed solutions to both problems in each of three languages. Each composition task was allocated a maximum time of five minutes. The order of languages in which the students constructed solutions was randomly varied to distribute the effects of the programmer gaining experience from previous solution attempts.

Performance was determined by the correctness of the student solutions. Language technicalities such as syntax errors or missing declarations were ignored. Each problem solution in each language was categorized into one of four categories:

- completely correct - the solution was complete and correct

- nearly correct - correct logic was employed, but solution had minor problems such as incorrect initialization
- conceptually but not logically correct - concepts were applied correctly, but program solutions had significant logic errors
- incorrect - incorrect concepts and logic

The programs were graded by both authors separately, with all inconsistencies being resolved together. The results were then tabulated and analyzed.

4: Results

At first, we considered assigning numeric scores as a measure of the correctness of each solution. However, following the recommendations of [5], we instead categorized the solutions into four correctness groups. The temptation to use an absolute numeric scale was strong, because the strongest statistical analysis can only be performed in the case of an evaluation scale that is either interval (numbers with meaningful differences, as well as equality and order) or ratio (interval scale plus meaningful ratios). But because the interval and ratio scales require the ability to make numerical computations based on the program scores ("is the difference between solutions in categories 1 and 2 the same as the difference between solutions in categories 2 and 3?"), they could not be used in our experiment. This lead to our choice of an ordinal "categories of correctness" scale as our measuring device. The ordinal scale can be used when only equality and order are present. (The last and least powerful possibility is the nominal scale, which requires only equality within a category.) Use of the ordinal scale dictated use of Chi-square (χ^2) statistical analyses, which is sufficient to determine statistical relevance.

In the following section, we present some of the more pertinent results, with complete results being reported in [11]. Recall that all of the subjects were familiar with either the textual language C or Pascal, none were familiar with Forms/3, and only one was familiar with APL.

4.1: Overall results

When the results of the two problems attempted by the subjects are combined, the dominance of Forms/3 is readily apparent, as shown in Table 1. Further breakdown of results of each problem indicate Forms/3 and OSU-APL statistically held an advantage over Pascal in one of the two problems, and Forms/3 and Pascal statistically outperformed OSU-APL in the other problem. These results will be presented in more detail in the next sections.

	completely correct	nearly correct	conceptually but not logically correct	incorrect
Pascal	45	6	25	44
Forms/3	88	9	9	14
OSU-APL	64	6	8	42

Table 1: Combined results.

4.2: Problem 1

The first problem was to append two appropriately shaped matrices together. The problem statement is given in figure 2a, and the solutions are given in figures 2b-d. We did not know what to expect for this problem. We feared it might be too simple, causing all subjects to get all correct answers in all three languages. The results are shown in table 2. We were surprised to find that so many of the students could not correctly solve this problem in Pascal in the 5 minutes given.

	completely correct	nearly correct	conceptually but not logically correct	incorrect
Pascal	7	1	21	31
Forms/3	53	0	2	5
OSU-APL	49	3	2	6

Table 2: Results for Problem 1.

Chi-square analysis indicates a statistically significant difference in the Forms/3 results versus the Pascal results (significant at the 0.005 level), but not in the Forms/3 performance versus the OSU-APL performance. These results convey the difficulty of manipulating matrices in traditional textual languages such as Pascal. Many of the Pascal solutions were conceptually but not logically correct, yielding solutions like those shown in figure 2e.

The illustrations in figure 2e also serve to highlight the ease with which matrix problems can be solved visually, since the correct illustration is in fact almost the same as the Forms/3 solution. It is interesting to note that the Forms/3 performance on this problem was approximately the same as the OSU-APL performance, in light of the fact that there is a built-in OSU-APL operator that appends two matrices which had been presented during one of the examples.

Write a program in Pascal that appends two appropriately shaped matrices into a single matrix For example, the program applied to the matrices

```
1   2   3            4   5   6
6   7   8     and    9   10  11
12  13  14           15  16  17
```

returns the matrix

```
1   2   3   4   5   6
6   7   8   9   10  11
12  13  14  15  16  17
```

Figure 2a. Problem statement for matrix append, the first problem on the experiment. There were three such problems, one for each language. The order of these three questions was varied randomly, in order to eliminate question sequencing as a factor.

```
procedure append
      (a: Matrix1; b: Matrix2;
      R1, C1, C2: integer;
      var c: Matrix3);
var i, j: integer;
begin
      for i := 1 to R1 do
      begin
         for j := 1 to C1 do
            c[i,j] := a[i,j];
         for j := 1 to C2 do
            c[i,j+C1] := b[i,j];
      end;
end;
```

Figure 2b. Matrix append in Pascal

```
result = Append (C, m1, m2)
```

Figure 2c. Matrix append in OSU-APL. There is a built-in Append operator that allows this one-line solution in OSU-APL.

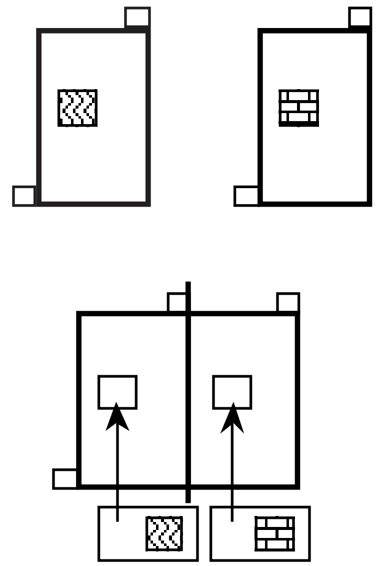

Figure 2d. Matrix append in Forms/3. The top two matrices are to be combined. The user places a new matrix on the form (second row), divides it into two regions, and defines the formula for a representative cell in each region. For example, the formula for cells in the left region is a reference to the corresponding cells in the top left matrix.

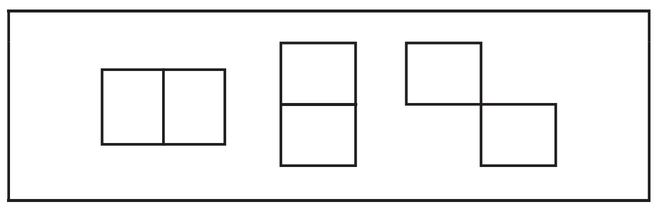

Figure 2e. The results of appending matrices should yield the results shown on the left. Instead however, many of the Pascal solutions produced the results shown in the center and on the right. Some solutions like the center drawing were also created in OSU-APL.

4.3: Problem 2

The second problem was intended to be more difficult, probing more deeply into the subjects' understanding. The problem was to produce the first N Fibonacci numbers. See figures 3a-3d. Because recursion was not included for any of the languages in this study, the Pascal solutions were either the traditional iterative solution using two variables, or the vector-based solution shown in the figure. The Forms/3 and OSU-APL solutions were vector-based.

On the Fibonacci problem, Forms/3 and Pascal performance was very similar, and both out-performed OSU-APL. Chi-square analysis indicates a statistically significant difference between the Forms/3 performance and the OSU-APL performance (significant at the 0.005 level), and no statistically significant difference between the Forms/3 results and the Pascal results. We found the latter surprising, because we had expected superior performance on this problem in Pascal since most students

Write a program in OSU-APL that computes the first N elements of the Fibonacci sequence. In the Fibonacci sequence, the first two elements are 1, and each element after that is the sum of the previous two elements.

For example, given an N of 10, the program produces:
1 1 2 3 5 8 13 21 34 55

Figure 3a. Problem 2 statement. Like Problem 1, this problem was stated three times, once for each language, with the order varying randomly.

```
procedure fib
      (var v: Vector; n: integer);
var i: integer;
begin
      v[1] := 1;
      v[2] := 1;
      for i := 3 to n do
         v[i] := v[i-1] + v[i-2];
end;
```

Figure 3b. A Pascal solution to the Fibonacci sequence.

```
v = 1 1
temp1 = Take(2, R, v)
temp2 = RowReduce(+, temp1)
v = Append (C, v, temp2)
if Dimension(C, v) <> N then 2
```

Figure 3c. An OSU-APL solution to the Fibonacci sequence.

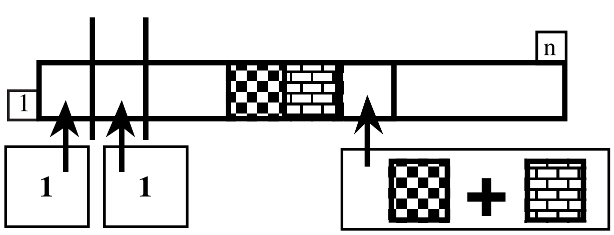

Figure 3d. A Forms/3 solution to the Fibonacci sequence.

had prior experience with this problem in Pascal. (A post-test given to all subjects found that all except 6 of the subjects had either seen or written Fibonacci in Pascal or C prior to the study.) The results of this problem are summarized in Table 3.

	completely correct	nearly correct	conceptually but not logically correct	incorrect
Pascal	38	5	4	13
Forms/3	35	9	7	9
OSU-APL	15	3	6	36

Table 3: Results for Problem 2.

5: Discussion

While designing this experiment, we deliberately chose not to have subjects explain previously-written programs. We believe that explaining previously-written visual programs statically on a piece of paper cannot accurately measure usability or comprehensibility of a visual programming language whose syntax is at least partially dynamic. Instead, it measures understandability of an artificially limited representation of a visual program compared to the full syntax of a textual language. We do not consider this to be a valid comparison. To minimize this problem, we chose to measure only program constructibility.

The immediate visual feedback feature of Forms/3 was not present in this experiment. Also absent were any advantages that would be gained by the fact that APL is traditionally interpreted. If on-line experimentation was attempted, it would allow for the measurement of time to create correct solutions, i.e., to incorporate ease of debugging as an evaluation criterion.

One difficulty in this experiment was how to statically describe a visual programming language to a new audience in sufficient detail to allow the subjects to acquire the necessary programming skills. Initial attempts at performing this task via static presentation methods were unsuccessful. Since an important attribute of the visual programming language is the process by which users program in it, we attempted to solve this problem by simulating the use of the language via gradually uncovering transparencies and similar techniques. Another difficulty encountered in earlier presentations may have been the distraction caused by the audience's mental efforts in formulating the solution in other languages with which they were familiar. We were able to address this problem by first presenting the solution to the example in Pascal, so that the audience felt comfortable in moving on to other approaches. We also worked to keep the subjects' interest by adopting an example-driven approach.

It is interesting to consider the types of programming errors made in the different languages. In all three languages there were students who simply had no idea what approach to take, and students who took an approach that was flatly wrong (added the wrong things together, etc.) But errors found only in the Pascal solutions were lack of generality caused by incorrect constants and size variables, and lack of time or patience to complete the detailed instructions needed (replaced by phrases like "etc., etc." or more colorfully "hack, hack"). Pascal also had a number of incorrectly constructed matrices as previously discussed in figure 2e. Unlike the other two languages, in Pascal many errors were related to mechanics–incorrect nesting of loops, subscripting computations, initializations, and prematurely destroying values stored in temporary variables. (Syntax and declaration errors were ignored.) In OSU-APL, although there were several errors having to do with subscripting computations and incorrect use of loops, errors often centered on not being able to devise the needed combinations of matrix manipulations. Errors particular to Forms/3 centered on not understanding how the multiple regions of a matrix could be used.

6: Conclusions

We have presented empirical evidence that supports the claim that matrix manipulation programs can be more easily constructed in the visual language Forms/3 than in the textual languages Pascal and OSU-APL. This result came about in spite of the difference in programmer experience in the different languages used in this study. The matrix and vector manipulation tasks were so strongly amenable to visual programming that programmers were more successful in constructing programs for tasks in the newly-learned visual programming language Forms/3 than in the well-known textual language Pascal or in the simplified textual matrix language OSU-APL.

To our knowledge, this is the first empirical study that has been done comparing constructibility of programs in visual versus textual languages. Although this study was small, measuring only one small problem domain, we look forward to using the insights and experience gained to plan more elaborate studies testing the use of visual languages for more general problem-solving tasks.

Acknowledgments

We would like to express our sincere appreciation to Curtis Cook for his many helpful suggestions influencing our design and analysis of this experiment, and to the students of CS 381 during Winter 1993 at Oregon State University for their enthusiastic participation in the study.

References

[1] B. Bell, J. Rieman, and C. Lewis. Usability Testing of a Graphical Programming System: Things We Missed in a Programming Walkthrough. CHI '91, New Orleans, 7-12, Apr. 27-May 2, 1991.

[2] R. E. Brooks. Studying Programmer Behavior Experimentally: The Problems of Proper Methodology. Communications of the ACM, 23(4), 207-213, Apr. 1980.

[3] M. Burnett. Abstraction in the Demand-Driven, Temporal-Assignment, Visual Language Model. Ph.D. Thesis, University of Kansas, Aug. 1991.

[4] M. Burnett and A. Ambler. A Declarative Approach to Event-Handling in Visual Programming Languages. IEEE Workshop on Visual Langs., Seattle, 34-40, Sept. 1992.

[5] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. Software Engineering Metrics and Models. Benjamin Cummings Publishing Company, Menlo Park, CA, 1986.

[6] N. Cunniff and R. P. Taylor. Graphical vs. Textual Representation: An Empirical Study of Novices' Program Comprehension. Empirical Studies of Programmers: Second Workshop, (G. Olson, S. Sheppard, E. Soloway, eds.), Ablex Publishing, Norwood, NJ, 1987.

[7] E. P. Glinert, M. M. Blattner, and C. J. Frerking. Visual Tools and Languages: Directions for the '90s. 1991 IEEE Workshop on Visual Langs., Kobe, Japan, 89-95, Oct. 8-11, 1991.

[8] T. Green, M. Petre, and R. Bellamy. Comprehensibility of Visual and Textual Programs: A Test of Superlativism Against the 'Match-Mismatch' Conjecture. Empirical Studies of Programmers: Fourth Workshop, (J. Koenemann-Belliveau, T. Moher, S. Robertson, eds.), Ablex Publishing, Norwood, NJ, 1991.

[9] K. E. Iverson. A Programming Language. Wiley, 1962.

[10] K. Jensen and H. Wirth. Pascal User Manual and Report, Third Edition. Springer-Verlag, New York, 1985.

[11] R. Pandey and M. Burnett. Is It Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study. TR 93-60-08, Oregon State University, Mar. 1993.

[12] J. Stasko, A. Badre, and C. Lewis. Do Algorithm Animations Assist Learning? An Empirical Study and Analysis. Proc. InterCHI '93, 1993.

[13] G. Viehstaedt and A. Ambler. Visual Representation and Manipulation of Matrices. Journal of Visual Langs. and Computing, 3(3), 273-298, Sept. 1992.