End-User Programming as Translation: An Experimental Framework and Study

Christopher Hundhausen, Ravikiran Vatrapu, and Joshua Wingstrom
Laboratory for Interactive Learning Technologies
Department of Information and Computer Sciences
University of Hawai'i
Honolulu, HI 96822 USA
{*hundhaus, ravikira, wingstro*}*@hawaii.edu*

Abstract

One of the reputed advantages of end-user programming languages is that they support a given problem domain with a set of programming abstractions that are "just right" for the end-users who need to program within that domain. Green and Petre's [1] Cognitive Dimensions Framework accounts for this advantage in terms of the "closeness of mapping" dimension: the closer the programming domain is to the problem domain, the easier the programming task. This suggests that programming might be conceptualized as a process of translation, with "close" translations being more efficient and error-free than "distant" ones. While there appears to be a strong intuitive basis for this view, there presently exists little direct empirical evidence. To that end, we present an experimental framework for systematically exploring the "closeness of mapping" dimension within the programming domain of college-level textbook algorithms and data structures. A pilot study we conducted within that framework provides preliminary evidence in support of one of our hypotheses.

1. Introduction

Inspired by instructional model used in architecture, the "Algorithms Studio" project [2] is exploring a studio-based approach to integrating algorithm visualization technology into second- and third-semester college computer science courses on algorithms and data structures. In this approach, students construct their own, "low fidelity" (rough, unpolished) visualizations of algorithms under study, and present those visualizations to their peers and instructor for feedback and discussion at regularly scheduled critique sessions.

To support this approach, we have developed ALVIS [3], a prototype low-fidelity visualization environment. We established two general usability goals for ALVIS:

1. ALVIS users should be able to create a visualization as quickly and accurately as they can program the corresponding algorithm in Java (the programming language they already know).
2. ALVIS users should be able to create a visualization as quickly and accurately as they can construct a "storyboard" using simple art supplies (the medium upon which ALVIS's conceptual model is based).

Given these goals, it became clear to us that, in order to verify the usability of ALVIS, we actually needed to empirically compare ALVIS against Java and art supply storyboards. However, designing a study to compare the programming efficiency and accuracy promoted by alternative programming environments proved to be more difficult than we anticipated. In deciding on a notation, we observed that certain descriptive notations for algorithms would appear to match certain programming notations better than others, potentially setting up programming tasks that are faster and more accurate. Note that, within Green and Petre's Cognitive Dimensions Framework [1], the "closeness of mapping" dimension would predict such differences: the fewer "programming games" that need to be played in order to map from the problem domain to the programming domain, the easier the programming task will be.

In this TechNote, we present our foray into empirically validating the "closeness of mapping" dimension within the specific programming domain of textbook algorithms and data structures. We begin by proposing a series of hypotheses related to the "closeness of mapping" dimension within this domain, along with a general experimental framework for testing those hypotheses. We then briefly present a pilot experiment that provides preliminary evidence in support of one of our hypotheses. A fuller treatment of this work can be found in [4].

2. Experimental Framework

Our focus is on the general task of translating alternative descriptions of "textbook" computer algorithms into programming languages. We hypothesize that the efficiency and accuracy with which that translation task can be performed depend on the goodness of the match between (a) the descriptive notation of the algorithm, and (b) the target programming language.

With respect to (a) descriptive notations, we are interested in algorithms typically studied in second- and third-

semester college courses on algorithms and data structures. A survey of representative textbooks suggests that three alternative notations are typically used to describe such algorithms:

1. *Natural Language (NLA)*—a high-level description of how the algorithm operates;
2. *Pseudocode (PSE*)—a description of the algorithm in a language that closely resembles a programming language; and
3. *Visual Examples (VIS)*—a series of data structure pictures that illustrate the way in which the algorithm operates on a sample data set.

With respect to (b) target programming languages, we are interested in three competitors within the context of our development of ALVIS [3], an end-user language and environment for algorithm visualization:

1. *Java* (JAV)
2. *SALSA, ALVIS's underlying scripting language* (SAL), and
3. *Art Supply Storyboards* (ART).

2.1 Hypotheses

Based on a "closeness of mapping" analysis [1] of the of the three descriptive notations vis-à-vis the three programming environments, we propose the following six hypotheses:

(H1) Translations from PSE: JAV >> SAL ≈ ART. Pseudocode and Java would appear to be well-matched, because they employ similar programming abstractions.

(H2) Translations from VIS: ART >> SAL >> JAV. The visual example descriptive notation would appear to best match art supply storyboards; indeed, both notations depict the algorithm in terms of a visual representation operating on a sample input data set. Likewise, because an ALVIS storyboard representation accompanies a SALSA program, the visual example notation would appear to match SALSA more closely than it matches Java.

(H3) Translations from NLA: JAV ≈ SAL ≈ ART. Because it lacks both a visual representation and a set of formal abstractions, natural language does not appear to match any of the target programming notations particularly well.

(H4) Translations to JAV: PSE >> NLA >> VIS. Java would appear to be well-matched with pseudocode, because both notations make use of similar programming abstractions. In contrast, Java would appear to be a relatively poor match with natural language, which is less formal and precise than Java. In addition, the visual examples notation would appear to be even more of a mismatch, owing to the fact that it is not only less formal, but also more concrete (it illustrates a specific input data set rather than. general input) and more visual.

(H5) Translations to SAL: VIS ≈ PSE >> NLA. SALSA, and its accompanying ALVIS visual representation, would appear to occupy a middle ground between pseudocode and visual examples. Like pseudocode, SALSA is procedural, and employs similar iterative and conditional programming abstractions. Like visual examples, SALSA's visual representation (an ALVIS storyboard) depicts the algorithm operating on a sample data set.

(H6) Translations to ART: VIS >> PSE ≈ NAL. As argued in H2, visual examples would appear to be an excellent match with art supply storyboards. In contrast, both pseudocode and natural language would appear to be poorly matched with art supply storyboards, for two reasons: (a) they describe an algorithm in general terms, as opposed to with respect to a specific example; and (b) they are purely textual, as opposed to visual.

2.2 Design of General Experiment

To test these hypotheses, we adopt a 3 × 3 mixed-factor design with nine conditions. We obtain these conditions by crossing the two independent variables discussed above:

1. *Descriptive notation*—pseudocode (PSE), visual examples (VIS), and natural language (NLA).
2. *Programming environment*—Java (JAV), SALSA (SAL) and art supply storyboard (ART).

Assuming 12 participants per condition, Table 1 presents this design in matrix form.

We propose two dependent variables for measuring the effects of these independent variables:

1. *Programming speed*—The time to complete a programming task
2. *Programming accuracy*—The extent to which a programming solution is free of syntactic and semantic errors.

		Programming Environment		
		JAV	SAL	ART
Descriptive Notation	NLA	12	12	12
	PSE	12	12	12
	VIS	12	12	12

Table 1. 3 × 3 Mixed-Factor Experimental Design

3. A Preliminary Experiment

As an initial step toward verifying the plausibility of our hypotheses, we conducted a pilot experiment that compared two strategically-chosen conditions in the framework: PSE × JAV vs. VIS × JAV.

We chose these two cells of the matrix (see Table 1) for two reasons. First, H4 predicts significant differences between these two cells: Pseudocode is predicted to be the best match with Java, whereas visual examples are predicted to be the worst match with Java. Second, we felt that, at this initial stage of the research, our study would have the best chance of avoiding design flaws by manipulating just one of the two independent variables.

3.1 Participants, Materials, and Tasks

We recruited 16 participants (7 female, 9 male) out of the spring semester, 2003 offerings of ICS 211 ("Computer Science 2") and ICS 311 ("Algorithms") at the University of Hawaii. Seven were enrolled in ICS 211; the other nine were enrolled in ICS 311. Participants were all majoring in information and computer science, and had an average self-reported cumulative grade point average of 3.3 (on a 4 point scale). Participants received a $20 honorarium for their participation in the study.

Participants used Borland® JBuilder® 8.0 to complete an "easy" and "hard" programming task. For the "easy" task, participants coded the "Find Max Value" algorithm. For the "hard" task, they coded the "Shuffle Left" algorithm: Given an array of integers, return the array with all zeros removed by successively "shuffling left" all array values that are to the right of any zeros.

3.2 Procedure

In a 30-minute screening session prior to their participation in the study, participants completed an informed consent form, background questionnaire, and programming pretest. In the main study session, which lasted around 60 minutes, participants (a) completed a 15 minute JBuilder® tutorial, and (b) performed the "easy" and "hard" programming tasks, along with accompanying questionnaires that had them rate task difficulty.

3.3 Results and Discussion

Tables 2 and 3 present our main results. As predicted by our hypothesis, a two-sample *t*-test (one-tailed) found that the PSE group completed the "hard" task signifycantly ly faster (t = -2.97, df = 15, $p < 0.0050$) and more accurately (t = 1.78, df = 15, $p < 0.0488$) than the VIS group. Our observations suggest that VIS participants required substantial amounts of time just to read through the visual example of "Shuffle Left," which spanned five pages. In contrast, PSE participants tended to scan the 16-line description of "Shuffle Left" only briefly before they began coding. We are reanalyzing our videotapes in order to see if the two groups differ signicantly with respect to "reading" time. If found, such a difference would shed further light on this result.

With respect to perceived difficulty, the VIS group found the "hard" task to be significantly more difficult to program (t = -2.25, df = 15, $p < 0.0409$). This result echoes the quantitative results, indicating that the participants themselves sensed the difference in difficulty.

	Condition	
Dependent variable	PSE	VIS
"Easy" Task		
$Time_{initial}$ (sec)	503.3 (112.7)	644.4 (361.7)
$Time_{final}$ (sec)	566.6 (126.9)	784.1 (442.8)
$Acc_{initial}$ (out of 11)	11.0 (0.0)	10.8 (0.7)
Acc_{final} (out of 11)	11.0 (0.0)	11.0 (0.0)
"Hard" Task		
$Time_{initial}$ (sec)	540.8 (92.4)	984.1 (411.4)
$Time_{final}$ (sec)	792.3 (302.7)	1494.6 (562.9)
$Acc_{initial}$ (out of 21)	18.1 (2.6)	16.4 (2.5)
Acc_{final} (out of 21)	20.1 (1.1)	18.9 (1.6)

Table 2. Mean Task Times and Accuracies by Condition (standard deviations are in parentheses)

	Condition	
Dependent variable	PSE	VIS
"Easy" Task		
Interpretation Difficulty	2.3 (1.6)	1.5 (0.8)
Programming Difficulty	2.5 (1.6)	3.6 (2.8)
"Hard" Task		
Interpretation Difficulty	4.1 (2.9)	6.5 (2.6)
Programming Difficulty	3.5 (2.1)	6.3 (2.7)

Table 3. Mean Subjective Ratings of Task Difficulty by Condition (standard deviations are in parentheses). *Ratings were on a scale of 1 (easiest) to 10 (most difficult).*

4. Acknowledgment

This work was supported by the National Science Foundation under grant no. 0133212.

5. References

[1] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: A 'cognitive dimensions' framework," *J. Visual Languages and Computing*, vol. 7, pp. 131-174, 1996.

[2] C. D. Hundhausen, "The "Algorithms Studio" Project," in *Proc. IEEE HCC '02*. Los Alamitos, CA: IEEE Computer Society Press, 2002, pp. 99-100.

[3] C. D. Hundhausen and S. A. Douglas, "Low fidelity algorithm visualization," *J. Visual Languages and Computing*, vol. 13, pp. 449-470, 2002.

[4] *C. D. Hundhausen, R. Vatrapu, & J. Wingstrom, "End-user programming as translation: An experimental framework and study," available at http://eecs.wsu.edu/~hundhaus/veupl/pub/hcc03f.pdf.*