The Impact of Human-Centered Features on the Usability of a Programming System for Children

John F. Pane
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 8078
pane+@cs.cmu.edu

Brad A. Myers
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
+1 412 268 5150
bam+@cs.cmu.edu

ABSTRACT

HANDS is a new programming system for children that was designed for usability. This paper examines the effectiveness of three features of HANDS: *queries, aggregate operations, and data visibility.* The system is compared with a limited version that lacks these features. In the limited version, programmers can achieve the same results but must use more traditional programming techniques. Children using the full-featured HANDS system performed significantly better than their peers who used the limited version. This provides evidence that usability of programming systems can be improved by including these features.

Keywords

End-user programming, psychology of programming, user studies, programming languages.

INTRODUCTION

The basis for generation and selection of the features of HANDS came from prior empirical research on programmers [1] and new studies conducted by the authors [2, 4].

Overview of the HANDS System

In contrast to the *von Neumann machine*, HANDS uses a familiar model to represent the computation: a *character* sitting at a *table*, manipulating *cards* that contain all of the program's data. The HANDS language provides operators that closely match the problem-solving methods observed in non-programmers. It uses an *event-based* style of programming, and provides *queries* and *aggregate operators* to allow more concise high-level expressions for tasks that require synthesis of many primitives in other languages [3].

The following sections describe the three features examined in this paper. The HANDS system was compared with a limited version of the system that lacks the features. In the limited version, the programmer must use the more traditional methods that are typically seen in popular programming systems.

Queries

HANDS provides a query mechanism that allows the programmer to assemble a list of objects on demand. For example, the query **all flowers** searches for all of the cards that contain the string "flower" and returns a list of their names (e.g., **rose, tulip, orchid**). HANDS supports more complex queries, but only this simple form was used in this study.

Most programming systems do not have a query feature. In those systems, the programmer must create and maintain data structures that provide access to the desired information. This is also necessary in the limited version of HANDS. For example, the programmer could create a card that holds a list of all of the flowers. This list would have to be updated each time a flower is added or removed from the program.

Aggregate Operators

In HANDS, all operations can be performed on a whole list of objects, including query results, with a single command. For example, **set nectar of all flowers to 0** will achieve the desired effect no matter how many flowers there are. Using the above example, the rose, tulip, and orchid cards would all have their nectar properties set to 0.

Most traditional programming systems do not support aggregate operations. In those systems, the programmer must iterate over the list of objects, operating on them one at a time. This is also the case in the limited version of HANDS, where the example shown above can be accomplished as follows:

```
with garden's flowerList calling each the flower
        set nectar of the flower to 0
end with
```

Visibility of Data

All data in HANDS is stored on cards, in name-value pairs called properties. Cards are always visible, even when the program is not running. They can be created and edited by direct manipulation as well as by actions taken by the program itself. The properties of multiple cards can be viewed simultaneously.

Traditional programming systems often do not provide these features for data. Variables might exist only temporarily while certain parts of the program are running. Data may not be visible to the programmer unless a debugging tool is used. In some systems, objects can only be created by executing code, and they do not exist when the program is not running. Systems that display the properties of objects may be limited to showing only one object at a time. The limited version of HANDS shows the properties of

Appeared in *CHI 2002 Extended Abstracts: Conference on Human Factors in Computing Systems*, Minneapolis, MN: ACM Press, April 20-25, 2002, pp. 684-685.

only one card at a time, and the other cards are not directly visible on the table.

THE STUDY

Participants

Volunteers were recruited from the fifth-grade class at a demographically-diverse public elementary school in Pittsburgh. The 23 volunteers ranged in age from 9 to 11 years old. There were 12 girls and 11 boys. All were native speakers of English, and none had computer programming experience. The participants came to the Carnegie Mellon campus on a Saturday morning for a three-hour session. 12 of the participants used the full-featured HANDS system (*Full*), and the other 11 used the limited system (*Limited*).

Materials

In the *Full* condition, a 13-page tutorial was used to teach the participants the basics of the HANDS system. The tutorial began with an empty program, and the participants built a program with several flowers and a bee that flies around collecting nectar from them. The bee is controlled by keyboard commands, and the program displays some basic statistics about the amount of nectar the flowers have and which flower has the least.

The tutorial for the *Limited* condition was derived from the full-featured tutorial. Those portions utilizing a feature that was missing in the limited system were replaced with material teaching the easiest way to use the system's remaining features to achieve the same result. This modification increased the size of the tutorial by one page, to 14 pages.

After completion of the tutorial, participants were given a two-page set of five tasks, plus a bonus problem. The solutions to each task would utilize at least one of the features that are missing in the limited system. All participants started the tasks by loading a partially implemented program. This program was similar to the one they had been working on, but it had more bees and some pre-defined cards to help solve the tasks. Once again, these materials were as similar as possible in the two conditions, differing only where necessary due to the limitations of the reduced-feature version of HANDS.

Procedure

The participants worked individually, at their own pace. When they finished the tutorial, they immediately started on the tasks. They could continue to refer to the tutorial while solving the tasks. Participants could stop working before the three-hour session was over if they finished the tasks, or if they wanted to quit for any other reason.

The experimenters answered questions and helped with any problems that the participants encountered, unless the assistance would reveal part of a task solution. In such a case, the experimenters simply referred the participants to material in the tutorial that might be helpful.

RESULTS

There was no significant difference in performance between boys and girls. In the *Full* condition, 75% (9 of 12) of the participants completed the tutorial and began to work on the tasks; while this ratio was 82% (9 of 11) in the *Limited* condition. This difference is not significant, and the remainder of this analysis examines only the participants who achieved this level of success.

On average, the participants in the *Full* condition spent 121 minutes working on the tutorial, while the participants in the *Limited* condition spent 139 minutes. This difference is not significant, but it does mean the participants in the *Full* condition had more time remaining to complete the tasks. Indeed, on average the participants in the *Full* condition spent 36 minutes on the tasks, while those in the *Limited* condition spent 30 minutes. However this difference is also not significant.

Participants received one point for each task problem they completed correctly. No partial credit was given. With the bonus problem, the maximum score was 6. In the *Full* condition, seven participants scored 1 or higher. The scores ranged from 0 to 6, with an average of 2.1. In the *Limited* condition, only one participant scored received any points. This person scored 1, and the average of the group was 0.1. This difference in task performance is significant ($p<.05$).

Cumulatively, the participants in the *Full* condition solved 19 tasks in 323 minutes, at a rate of 6 minutes per solved task. The participants in the *Reduced* condition solved one task in 269 minutes, a rate of 269 minutes per solved task.

CONCLUSION

The superior performance of participants in the *Full* condition can be attributed to the presence of queries, aggregate operations, and data visibility in the system they used. This suggests that these features could improve the usability of programming systems in general. However, the study does not tease apart the contributions of the individual features. It also does not provide any evidence whether the HANDS system as a whole is better than other programming systems. These questions will be addressed in future work.

ACKNOWLEDGMENTS

Thanks to Albert Corbett, Leah Miller, and Bernita Myers, and all of the participants. This research is funded in part by the National Science Foundation under Grant No. IRI-9900452. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the National Science Foundation.

REFERENCES

1. Pane, J.F. and Myers, B.A. Usability Issues in the Design of Novice Programming Systems. Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, (Pittsburgh, PA, August 1996).
2. Pane, J.F. and Myers, B.A. Tabular and Textual Methods for Selecting Objects from a Group. *Proceedings of VL 2000: IEEE International Symposium on Visual Languages*. (Seattle, WA, September 2000), IEEE Computer Society, 157-164.
3. Pane, J.F., Myers, B.A., and Miller, L.B. Using HCI Techniques to Design a More Usable Programming System. *submitted for publication* (2002), http://www.cs.cmu.edu/~pane/handsdesign.html.
4. Pane, J.F., Ratanamahatana, C.A., and Myers, B.A. Studying the Language and Structure in Non-Programmers' Solutions to Programming Problems. *International Journal of Human-Computer Studies*, *54*, 2 (February 2001), 237-264.