Can Cognitive Modeling Improve Usability Testing and Rapid Prototyping?

Robert L. West (robert_west@carleton.ca)
Department of Psychology; Carleton University
Department of Cognitive Science, Carleton University
Ottawa, Canada

Bruno Emond (bruno_emond@uqah.uquebec.ca)
Department of Education; Université du Québec à Hull
Hull, Canada

Abstract

We argue that usability testing, as employed in the rapid prototyping cycle, could be improved by testing simulated users We briefly review some of the benefits that this methodology could offer and discuss one approach to building a simulated user using ACT-R to embody a GOMS-like memory structure

Rapid prototyping is frequently used to design interface systems for commercial software The goal is to rapidly iterate a cycle of creating, evaluating, and redesigning, to produce an intuitive, easy to use interface In this paper we discuss how usability testing, a popular means of evaluation for rapid prototyping, could be augmented by testing simulated users. The simulated user is not a new idea; it has its origins in GOMS modeling (Card, Moran, & Newell, 1983), and has also been explored using cognitive architectures (e g , Gray, 2000; Howes & Young, 1996) such as ACT-R (Anderson & Lebiere, 1998) and SOAR (Newell, 1990). However, not much attention has been paid to integrating simulated users into rapid prototyping and usability testing.

Usability testing involves having users perform tasks on an interface prototype to see how well it works The actual testing procedure is relatively quick and informal Usually, four to seven subjects are asked to do a series of tasks using the interface prototype While usability testing is effective, like any methodology, it is not without problems There are a number of different ways that simulated users could improve usability testing but, due to space limitations, we will focus only on the problem of not enough subjects. Using lower numbers of subjects increases the chance of not finding a problem or of overestimating or underestimating the likelihood of a problem occurring in the user population (see the following for experimental evidence consistent with this claim: Spool & Schroeder 2001; Kessner, Wood, Dillon, & West, 2001; Molich, Thomsen, Karyukina, Schmid, Ede, van Oel, & Arcuri, 1999). Rapid prototyping does not allow enough time to test large numbers of subjects, so one solution is to use simulated subjects. However, in order to do this, simulated users need to be created and validated. Here we describe our initial attempts to do this.

The goal of this project is to develop a system that will improve the rapid prototyping process. However, developing a simulated user will not be much use if it is not accepted as a methodology within the usability domain. Related to this is the usability of the simulated user itself In rapid prototyping there is a very strong emphasis on developing prototyping tools that are fast and relatively easy to use. Therefore the simulated user should be as easy as possible to program and understand. A good model for this, we believe, is keystroke level GOMS (Card, Moran, & Newell, 1983), which is designed to be easy to use, easy to understand, and good enough for describing typical interface designs. The type of system we would like to create would be similar in nature to keystroke level GOMS but capable of exploring and learning how to use a novel interface Below we describe our initial attempts to create such a system using ACT-R.

For this type of system, issues concerning perception, attention, and action pose a problem. One solution would be to use an architecture designed to deal with these issues, such as ACT-RPM (Byrne & Anderson, 1998) or EPIC (Kieras & Meyer, 1997). However, in keeping with our goal of maintaining the simplicity of keystroke level GOMS, we assume that everything is accurately perceived so that the interface information can be coded (by LISP functions) into declarative memory as it becomes perceptually available. Perceptual, attentional, and motor operators are used only to estimate how long each action takes. Objects on graphical user interfaces tend to be fairly obvious and it is not commonly found that users fail to notice objects. However, the problem of not understanding the functionality of an object (e g , not understanding what an icon symbolizes) is quite commonly found in usability testing Unfortunately, it is problematic to model object recognition, even using ACT-RPM or EPIC. Therefore, object recognition would need to be assumed Note though, that it can also be assumed that an object is not understood to simulate the effect of this.

GOMS models have been shown to be both effective at modeling interface use and relatively easy and intuitive to grasp (John, 1995). We hypothesize that the reason for this is that the GOMS structure actually reflects the way humans organize their knowledge of how to use interfaces. From this perspective, trying to figure out an interface can be understood as the user attempting to mentally construct something similar to a GOMS model Based on this conceptualization, our approach has been to model the user as trying to construct a GOMS-like model of the task in their declarative memory system The template for how this information is built-up and stored in memory is based on a hierarchical structure similar to that used in GOMS (i.e., goals, methods, operators)

Figure 1 illustrates the template structure The rectangles represent goals, the boxes represent methods, and the circles

represent operators. The gray shapes represent the elements related to completing the first goal, and the white shapes represent the elements related to completing the second goal. At each level, knowledge of the sequence of steps is represented by chunks describing each step, with each chunk containing a slot that identifies the next step. The model uses the ACT-R goal stack to move to lower levels by pushing subgoals that get popped when the sequence at that level is complete. For example, if the goal was to open a file, a subgoal for a method of opening a file would be pushed. The first step in the method (e.g., "open file menu") would then push a sub-subgoal to execute the operators necessary for this step. The operators would fire in sequence, pop the sub-subgoal, and the system would move on onto the next step in the method. This process would repeat until the goal of opening a file is completed. The system would then move onto the next goal by popping the method goal. Note, that there can be more than one method sequence per goal, and more than one operator sequence per method. Learning which to use would be based on the chunk activation system. This would mean that the user would tend toward a "winner take all" strategy, increasingly preferring the methods used most in the past.

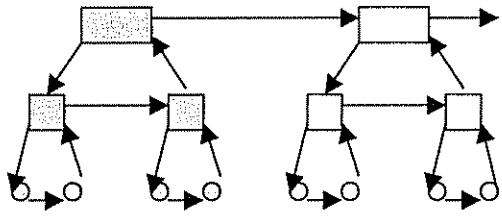

Figure 1. A template structure for storing interface operating knowledge

In a usability test, the trick is to provide the subject with enough information to understand the task, without revealing all of the steps involved. Thus, the subjects' task is to fill in the missing knowledge. In terms of the data structure described above this would be equivalent to a model of the task with some of the connecting chunks missing. In most cases these would be method chunks since the goal chunks would be needed to understand the task and the operator chunks would be based mainly on well known GUI objects (e.g., buttons, text fields, etc.). Based on this, we conceptualized the learning process as a search for missing method chunks. When the model reaches a gap in its knowledge it needs to search for chunks that it can link together to get to the next known step in the task. This process of building up chains of linked chunks to represent sequential actions is based on the scheme described by Lebiere & Wallach (1998) to account for sequence learning.

We plan to implement different strategies for searching for the missing chunks through production rules. As a starting point, we are considering four basic strategies. The first is choosing based on past associations from other interfaces. This would involve trying to achieve a goal by using methods that have been used successfully to achieve the same goal on other interfaces. This creates a problem in that it is necessary to provide the simulated user with a fair amount of domain knowledge, but it is also beneficial as it explicitly addresses the role of existing knowledge. The second strategy is to try interface objects that are labeled to indicate to the user that they are related to the goal (it would need to be assumed that the user understands the label, see above). The third strategy is to try interface objects at random to see how they transform the problem space, and the fourth strategy is to attempt to use the "help" features of the interface. However, this is only a starting point for further development through comparisons with human data. In terms of development, in our opinion, the best way to proceed is by using the open source code concept. We believe that this would best facilitate the process of development as well as the use of simulated users for rapid prototyping.

References

Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, NJ: Lawrence Erlbaum Associates.

Byrne, M. D. & Anderson, J. R. (1998). Perception and action. In Anderson, J. R. & Lebiere, C. (Eds.), *The atomic components of thought*. Mahwah NJ: Lawrence Erlbaum.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Gray, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science, 24(2)*, 205-248.

Howes, A., & Young, R. M. (1996). Learning consistent, interactive, and meaningful task-action mappings: A computational model. *Cognitive Science, 20*, 301-356.

John, B. E. (1995). Why GOMS? *Interactions 2* (10), 80-89.

Kessner, M., Wood, J., Dillon, R. F., & West, R. L. (2001). On the reliability of usability testing. *Proceedings of CHI 2001*. ACM, Seatle.

Kieras, D. E., & Meyer D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction, 12, 391-438.

Lebiere, C., & Wallach, D. (1998). Implicit does not imply procedural: A declarative theory of sequence learning. Paper presented at the *Forty First Conference of the German Psychological Society*, Dresden, Germany.

Molich, R., Thomsen, A. D., Karyukina, B., Schmid, L., Ede, M., van Oel, W., & Arcuri, M. Comparative evaluation of usability tests. *Human factors in computing systems CHI 99 Extended Abstracts*, 83-84, 1999.

Newell, A. (1990). *Unified theories of Cognition*. Cambridge, Mass: Harvard University Press.

Spool, J., & Schroeder, W. (2001). Testing web sites: Five users is nowhere near enough. *CHI 2001 Extended Abstracts*, 285-286.