Can Cognitive Modeling Improve Rapid Prototyping?[1]

Robert L. West
Department of Psychology
Carleton University
Ottawa, Ontario, Canada
robert_west@carleton.ca

Bruno Emond
Department of Education
Université du Québec à Hull
Hull, Quebec, Canada
emond@uqah.uquebec.ca

ABSTRACT
Rapid prototyping is an area in which cognitive modeling is not frequently used. We examine some of the reasons behind this and argue that cognitive modeling could play an important role. Specifically we argue that usability testing could be improved by testing simulated users. We review 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.

Keywords
cognitive modeling, rapid prototyping, usability testing ACT-R, GOMS

INTRODUCTION
Cognitive modeling is often employed in the design of systems destined to be used by well trained personal performing tasks in which errors could have serious consequences (e.g., military systems, aviation systems, nuclear power plants). Not surprisingly, due to the serious nature of these types of systems, the process of redesign often allows time for careful development and testing. In contrast, commercial software products are often aimed at an audience containing a relatively high number of non-experts. No formal training in how to use the product is expected and most users will not study the manual. Also, these systems generally have very short development times. The process most commonly used for developing interfaces under these conditions is *rapid prototyping*.

Rapid prototyping may involve a number of different techniques such as cognitive walk through, heuristic evaluation, and usability testing. The idea is to rapidly iterate a cycle of creating, evaluating, and redesigning the system interface. Cognitive modeling is not extensively used in rapid prototyping. In our opinion, this is due to three interrelated misperceptions: (1) that the process of cognitive modeling is necessarily slow, (2) that cognitive modeling is useful only for well learned, routine tasks, and (3) that cognitive modeling (as applied to interface design) is too low level (e.g., using Fitts Law to model mouse movements). These misconceptions are based on understanding cognitive modeling in terms of the type of projects that cognitive modelers typically work on rather than any inherent limits involved in cognitive modeling. (this list of misperceptions was distilled from a number of discussions concerning this project with usability professionals).

In this paper we describe how usability testing, an important element of rapid prototyping, could be improved by testing simulated users. The simulated user is not a new idea, it has its origins in the GOMS approach to interface design [4]. However, GOMS (a system for modeling users using goals, operators, methods, and selection rules) was designed to model well learned, routine behaviors, so it is not appropriate for modeling the process of learning to use a new interface (although it can be used to estimate the learning time, see [10]). We will first describe how using a simulated user could improve usability testing and then discuss our approach to building a simulated user using ACT-R [2].

USABILITY TESTING
Usability testing is generally acknowledged to be one of the more effective methodologies for rapid prototyping. This is because, of all the methods used in rapid prototyping, usability testing is the only one to test naïve users, similar in nature to the users the interface is intended for (i.e., they are not usability professionals). 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. Typically, four to seven users are asked to do a series of tasks using the interface prototype. The problems they

[1] Carleton University Cognitive Science Technical Report 2002-05
URL http://www.carleton.ca/iis/TechReports

encounter are noted down and sometimes there is a criterion set that specifies how many users need to encounter a problem before it is considered worth changing. Typically, usability testing succeeds in detecting usability problems that were undetected during the design process.

PROBLEMS WITH USABILITY TESTING

While usability testing is effective, like any methodology, it is not without problems. This is evident from studies such as [12] and [9], which show that, given the same interface, standard usability practices do not find the same sets of problems. We have divided our list of problems into two classes. The first is problems inherent in the experimental procedure:

1. Low numbers of subjects – Using lower numbers of subjects increases the chance of overestimating or underestimating the likelihood of a problem occurring in the user population. It is also a problem for large interfaces (e.g., extensive e-commerce sites) as there are so many different ways in which they could be used. Large numbers of subjects would be required to cover all the permutations (e.g., see [16])
2. Poor control over subjects' backgrounds – Usability testing generally does not involve a detailed examination of subjects' backgrounds, making it difficult to know how well the results will generalize to the population. This is especially important when a product is targeted at a particular group.
3. Motivational validity – Subjects in a study may not have the same motivation to overcome problems as someone who has paid money for a program or needs to use it at work.
4. Subject availability – Usability studies are often carried out in environments in which subjects are not readily available. This can result in an odd selection of subjects and/or in using the same subjects over and over again. These situations will aggravate problems 2 and 3 respectively.
5. Experimental bias – Bias can be introduced through the comments and attitude of the experimenter or through the experimental design. The extent of this problem will depend on the experience and skill of the experimenter.
6. Focus on initial experience – Usability testing does not tell you anything about the long term learning process, which is important for complex products.
7. Focus on a single, isolated interface – Usability testing does not consider the possibility that a user may be in the process of learning several interfaces (e.g., learning to use a suite of office products).

The second class involves problems of interpreting the results of usability testing:

1. Understanding why a problem occurred – Usability testing relies on subjects' self reports, which may not always accurately reflect why a problem occurred (e.g., see [14]).
2. Judging the severity of a problem – Usability testing provides no means of judging the severity of a problem other than how many subjects encountered the problem. Thus a problem that is trivial to overcome may be judged to be as serious as a problem that occurred the same number of times but is very difficult or impossible to overcome.
3. Judging the quality of the study – Given the problems raised above it should be clear that sometimes usability testing will provide misleading results. However, since there is no theory or cumulative body of knowledge employed when evaluating usability study results, it is difficult to judge the quality of the work. Also, follow up research is rarely done to see if usability testing actually did result in a better interface. The result of this is that it is easy to fake expertise.
4. Lack of theory – Usability studies identify problems in specific interface designs and in most cases that is all. Because usability tests are not tied to a theory of usability, they do not directly increase in our knowledge of how to build better interfaces by testing specific hypothesis.

SIMULATED USERS

As noted above, usability testing is an effective means of finding usability problems. Our point, however, is that usability testing would be better if the issues listed above could be addressed. To do this we propose that usability testing could be augmented through the use of simulated users, created through cognitive modeling. To illustrate how this could work we will first address the perceived problems with using cognitive modeling for rapid prototyping, and then go through the problems with usability testing and show how using simulated users can address these problems.

The first perceived problem was that cognitive modeling is too slow. Of course building a cognitive model from scratch would indeed be slow. A more sensible starting point would be to select a well tested cognitive architecture to use to build the simulated user, such as ACT-R [2] or SOAR [13]. In addition to using a cognitive architecture we propose that *cognitive templates* could also be used. Gobet and Simon [5] use the term *templates* to refer to preexisting structures in memory that allow domain specific information to be quickly and effectively coded. Vicente [17] uses *templates* to refer to preexisting, generic structures used to guide problem solving (see also [18]). We propose that it is appropriate to use templates describing how data is

stored and how problem solving occurs when modeling a user exploring a new interface within an operating system that they are familiar with (e.g., Windows, Mac OS, UNIX). Using templates it should be relatively easy to construct a model for testing a particular interface. In addition the model could be preloaded with knowledge of the operating system and of programs commonly used within the operating system. Under these conditions, we believe a model could be constructed very quickly. Of course the success of the model would depend on the validity of the architecture, the templates and the knowledge coded into it, but the development and testing of this aspect of the model would take place outside of the rapid prototyping domain.

The second misperception, that cognitive modeling is only able to capture well learned, routine tasks, does not deserve very much attention. Possibly this view arises from people who have only been exposed to GOMS modeling, which does not typically deal with novel situations or problem solving. At any rate, problem solving within a well structured domain, such as an operating system, is exactly what cognitive architectures such as ACT-R or SOAR do best. More specifically, an interface can be thought of as a means of moving through a problem space. From this point of view, trying to accomplish a task with a new interface is equivalent to searching for a path through a problem space.

The third misconception is that cognitive modeling is too low level. Generally speaking, the task structure in usability studies is conceptualized in terms of higher level goals, and issues concerning keystroke level actions are not explicitly considered. Thus from a usability perspective "open the file menu" would typify the lowest grain size of task analysis. Cognitive modeling systems used to model interface use, such as GOMS [4], EPIC [11], and ACT-R PM [3] tend to focus on a lower grain size. For example, "move hand to mouse, move eyes to file menu, move mouse cursor to file menu, click mouse." Usability studies are rarely concerned with exactly how long actions take, which is one of the main things you learn by using such precise operators. However, how long things take can be an important factor when making decisions about how to use an interface (e.g., see [6])) and is also important when comparing model results to human results. Because of this we believe that a simulated user should include keystroke level actions that take set amounts of time, similar to a GOMS model. However, since the simulated user would come with these actions already loaded and with an understanding of how to combine them to perform common tasks, in most cases dealing with this level of analysis would involve very little effort from the usability tester. For example, the simulated user would understand the necessary sequence of motor operators needed to click on a button so it would only need to be told to "click the button."

ADVANTAGES OF USING SIMULATED USERS

We believe that using a simulated user in conjunction with usability testing can significantly reduce the problems associated with usability testing. First we address problems inherent in the experimental procedure:

1. Low numbers of subjects – Using a simulated user provides you with access to an almost unlimited number of simulated subjects. Due to random chance, and depending on the situation, the same model may run into a variety of different problems across simulations. By using many simulations it should be possible to estimate the probabilities for encountering each type of problem in the user population.
2. Poor control over subjects' backgrounds – With simulated users you have an exact knowledge of simulated user's background. Also, it would be possible to test simulated users with different backgrounds to see the effects of previous knowledge.
3. Motivational validity – Simulated users are always fully motivated.
4. Subject availability – Simulated users are always available.
5. Experimental bias – Simulated users are not influenced by bias introduced through the comments and attitude of the experimenter. In terms of bias introduced through the experimental design (e.g., neglecting to test some aspect of the interface), simulated users would not change the situation.
6. Focus on initial experience – Simulated users can be used to explore long term learning within an interface system by having them interact with it as many times as necessary, and by having them progress to more complex tasks. This is particularly important for programs with multiple levels of complexity where an assumption is often made that the user will master the lower levels before moving onto the higher levels
7. Focus on a single, isolated interface – Simulated users can also be tested on more than one interface. This can address the question of the effects of learning one interface on learning another interface for a commonly associated task (e.g., programs from a suit of office tools, such as word processors, spreadsheets, email, etc.).

Next we examine how simulated users could help in terms of interpreting the results:

1. Understanding why a problem occurred – With a simulated user it should always be possible to tell

why a problem occurred by examining the model and the course of events in the simulation. Also it may be possible to gain insight into why a human subject experienced a particular problem by using model tracing [1], which would involve having a model step through the same steps as the human subject to see what it takes to make the model make the same mistake

2. Judging the severity of a problem – With a simulated user it should be possible to judge the severity of a problem by seeing how long it takes a simulated user to work around the problem. If the simulated user can't worked around the problem, severity can be judged by seeing the extent to which information needs to be added to the model in order to overcome the problem.
3. Judging the quality of the study – Having a model that explains and extends the results of a usability study also provides a means to evaluate the quality of the study. Specifically, if getting the model to agree with the human results involves adding a lot of unrealistic assumptions to the model or involves excessively altering the background knowledge or the template structure of the model, then, provided the model has worked well under similar conditions, we should suspect a problem with the study. Also, a well developed model shows that the usability tester thoroughly analyzed the interface before designing the study.
4. Lack of theory – Building and testing models is a way of testing the theories embodied in the models. As better models are developed our understanding of what makes a good interface will increase.

LIMITATIONS OF SIMULATED USERS

As illustrated above, cognitive modeling has a lot to offer the rapid prototyping process. However, cognitive modeling is not a panacea. Some current limitations are listed below:

1. Usability - In rapid prototyping there is a very strong emphasis on developing prototyping tools that are fast and relatively easy to use. Therefore a simulated user should be as easy to program and to understand as possible. This is not the case with many cognitive modeling systems, especially those complex enough to create a simulated user. Thus usability is an important issue for developers of cognitive modeling systems. Currently this is a significant barrier to the use of cognitive modeling for rapid prototyping.
2. Perception - Another limitation of cognitive modeling is in perceiving graphical objects, such as icons. Recently there has been a focus on providing cognitive architectures with perceptual abilities. ACT-RPM [3] and EPIC [11] are notable in this regard. However, these systems still need to be told what graphic objects on the screen represent. So at this point, it is problematic to use cognitive modeling to ascertain how objects, such as icons, will be interpreted. Although, cognitive modeling can be used to examine the effect of understanding, or not understanding, the meaning of an object, by simulating both scenarios.
3. Bottom up attention - Cognitive modeling is currently limited in terms of accounting for bottom up attention. Therefore, given an interface, cognitive modeling will not tell you which elements will grab the user's attention first. This is especially true for interfaces with complex graphics, such as found on some web pages.
4. Unresolved theoretical issues – cognitive modeling is an active field of research and there are often competing theories about best way to model different cognitive abilities. So the best way to construct a simulated user may not always be clear.
5. Esthetics – cognitive models will not tell you much about how good an interface looks

However, these problems can actually be overcome by using usability testing. For (1), usability testing could be very helpful in making cognitive modeling systems easier to use. For (2) and (3) usability testing can be used to provide some idea about whether or not graphical objects can be understood and which objects tend to grab peoples' attention. This interface specific information can then be added to the simulated user. Usability testing also provides an excellent domain for testing different cognitive architectures. It is an extremely well defined domain with no shortage of people who are familiar with it. Thus usability testing could help problem (4) by providing a lot of valuable data for evaluating different modeling theories. Issues concerning individual differences can also be investigated as it should be possible to model individual usability results by using parameter fitting (we do not have a solution for problem 5).

Further details on how a simulated user could be constructed would depend on the specific project. So, as an example, we will give an overview of a simulated user that we have been working on.

EXPLORING AND LEARNING

One of the most important issues for modeling usability testing is conceptualizing the learning and exploration process. Our approach, since we specifically want to combine modeling with usability testing, has been to develop a conceptualization that makes clear linkages between the structure of the model and the process of usability testing. To do this we combined the conceptual structure for storing and accessing information from

GOMS with the learning and problem solving architecture of ACT-R. GOMS models have been shown to be both effective at modeling interface use and relatively easy and intuitive to grasp [8]. 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 mental 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 the hierarchical structure used in GOMS, but is also tied to the process of usability testing. The elements of the template are described below:

1. Goals – goals represent the set of sub goals that need to be achieved in order to complete the task. In usability testing it is necessary to provide the human subjects with goals or sets of goals to achieve. The simulated user should be provide with the same set of goals needed for the human subjects. As in usability testing, the trick is to provide enough information to allow the user to do this task, without revealing everything that the user needs to know to do the task. So within this framework, goals represent information that would be provided or can be assumed to be known by human subjects in a usability test
2. Methods – methods are fixed sets of steps that can be used to achieve goals. In a good usability test, human subjects should not be told the methods, unless certain methods would be known to the intended user but not to the test subjects. The same should be true for the simulated user.
3. Operators – operators are the specific actions, (physical, perceptual, or cognitive) needed to complete each step in a method. In human subjects, operators are characterized by taking a certain amount of time and sometimes by the chance of making an error. The simulated user should have these built in.

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 as 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. Essentially, methods are subgoals of task goals, and operators are subgoals of methods. For example, if the goal were 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 method step. The operators would fire in sequence, pop the sub-subgoal and move on onto the next step in the method. This process would repeat until the goal of opening a file is completed, after which the system would move onto the next task 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 act-r chunk activation system (from a GOMS point a view this would constitute learning the selection rules).

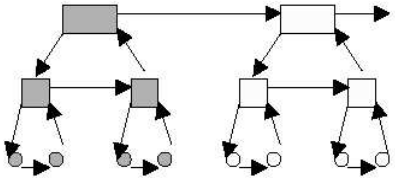

Figure 1. A template structure for storing interface operation knowledge

In a usability test, the user is missing information about the task. 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. Since we have defined the goals as being known and we also assume the user knows the operator sequences needed for basic navigation (e.g., how to use a mouse, how to use the keyboard), the missing chunks would be method steps. Based on this we conceptualized the learning process as a search for the missing method steps. Figure 2 illustrates search problem. The white boxes show a missing method (e.g., the user does not know how to use the interface to save a file). In this case the user must try different things and look for a sign (e.g., a label) indicating they are on the right track. The gray boxes show a missing method step (e.g., the user opens the menu to get to the "save" dialogue box but there is no menu item labeled "save"). In this case the user must decide whether to search for the next step (e.g., look for another menu item that could take you to a "save" dialogue box) or abandon this method and try a different one (e.g., look for a "save" icon). Notice that in order to do these tasks the simulated user needs a fair amount of knowledge, including a lexicon for understanding labels (e.g., see [7]) and knowledge about how interfaces tend to work (this could be generalized knowledge and/or specific knowledge of how other interfaces work).

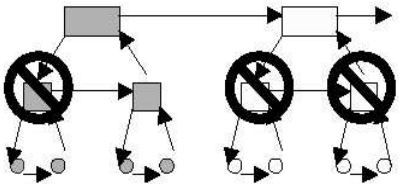

Figure 2. Representing a usability testing scenario in the template structure

We are implementing different strategies for searching for the missing chunks through production rules (i.e., if…then 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. To do this it is necessary to provide the simulated user with the same relevant domain knowledge that a human user would be expected to have. The second strategy is to try interface objects that are labeled to indicate to the user that they are related to the goal. 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.

LOW FIDELITY PROTOTYPING

Developing simulated users is only half the problem. The simulated users still need an interface prototype to interact with. The prototypes used in usability testing range from high fidelity to low fidelity. High fidelity prototypes are fully functional or almost fully functional interfaces. They have the advantage of providing human subjects with a realistic experience but have the disadvantage of being time consuming to develop. Low fidelity prototypes are mockups that have limited functionality. The advantage of low fidelity prototypes is that they are fast and cheap, and therefore very useful for testing potential interface designs early in the design process. Currently there are several projects to get simulated users to interact with relatively high fidelity interfaces (e.g., see [15]). The idea is to develop systems that allow simulated users to interact with the same software that human subjects interact with. This approach is good for high fidelity prototyping but may not be the best choice for low fidelity prototyping.

Because we are interested in testing simulated users early on in the rapid prototyping process, we have focused on low fidelity prototyping. To be useful, low fidelity prototyping systems must be relatively quick and easy to use. They also need to capture the elements of the full interface design that drive the way that human users interact with it. If completely successful in this regard, a low fidelity prototype is just as effective as a high fidelity prototype for testing human subjects. However, there is no way to know if you have succeeded without also building a high fidelity prototype and testing to see if people behave the same way with it. The same is not true for simulated users. Since we know how a simulated user works, we can know what aspects of an interface prototype will affect it and what will not. Also, we know that a simulated user will not be affected by the realism of the experience. In fact, a simulated user does not require a visual interface at all, it just needs to be told what is there and what are the effects of its actions.

To explore the possibilities offered by testing simulated users on low fidelity prototypes we created a prototyping tool called SOS-1.0 (Simple Operating System, Version 1). SOS-1.0 does not provide a visual representation of an interface. It is a system for constructing an abstract representation of an interface that a simulated user created with ACT-R 4 can interact with. SOS-1.0 is designed to investigate how knowledge and strategy drive the exploration of typical Windows interfaces. It assumes that simulated users can see what's on the screen, can find objects on the screen, and can activate objects. This amounts to an assumption that the interface is reasonably well designed from a visual point of view. The SOS-1.0 system is composed of four elements:

1. Screen - The screen is a container for what is visible at any given time
2. Windows - Windows can be opened or closed but they cannot occlude each other. If a window is open it is assumed to be visible.
3. Frames - Windows can be divided into frames. Frames are frequently used in GUIs to indicate shared functionality. For example, menu items are usually grouped within a frame. Thus frames can provide important clues to the simulated user
4. Objects - Objects can perform actions (e.g., buttons, scroll bars) or provide information (e.g., text, graphics) or both (e.g. icons, labeled buttons)

Menus and other structures are constructed. For example, a menu bar would be a frame containing labeled buttons that open windows (representing drop down menus) containing more labeled buttons (representing the menu items). The fact that dropdown menu boxes occlude the underlying window can be modeled by closing the underlying window and reopening it when the menu box is closed. Currently we are involved in using SOS to run simulated subjects, created in ACT-R 4, to evaluate the system. We are also working on a version of SOS that

will run with ACT-R 5 (ACT-R 5 has the perceptual /motor model from ACT-R PM built into it).

Our goal is to create an easy to use, low fidelity prototyping system. However, SOS also embodies a theory of cognition that says that people interpret standard interfaces according to a limited set of elements from which objects are mentally constructed. As noted above, to the extent that low fidelity prototyping systems capture the essential elements that drive the user, they will produce accurate results. Thus the use of low fidelity prototypes for testing simulated users or human users always embodies a cognitive theory of interface use (whether or not the usability tester is aware of it).

CONCLUSIONS

We believe that usability testing can be significantly improved through the use of simulated users. However, we do not believe that simulated users should be used to replace human usability testing. Rather, we suggest that the two techniques complement each other.

REFERNCES

1. Anderson, J. R., Kushmerick, N., & Lebiere, C. (1993). Navigation and conflict resolution. In Anderson, J. R. (Ed.), *Rules of the mind*, Erlbaum, Hillsdale, N.J.
2. Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Mahwah, NJ: Lawrence Erlbaum Associates.
3. 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.
4. Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.
5. Gobet, G., & Simon, H. A. (2001). Five seconds or sixty? Presentation time in expert memory. Cognitive Science, 24(4), 651-682.
6. Gray, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science, 24(2),* 205-248.
7. Howes, A., & Young, R. M. (1996). Learning consistent, interactive, and meaningful task-action mappings: A computational model. *Cognitive Science, 20*, 301-356.
8. John, B. E. (1995). Why GOMS? *Interactions. 2* (10), 80-89.
9. Kessner, M., Wood, J., Dillon, R. F., & West, R. L. (2001). On the reliability of usability testing. *Proceedings of CHI 2001*. ACM, Seatle.
10. Kieras, D. E. (1997). A guide to GOMS model usability evaluation using NGOMSL. In helander, M., Landauer, P., & Prabhu, P. (Eds.), *Handbook of human computer interaction*. Elsevier Science
11. 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.
12. 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.
13. Newell, A. (1990). *Unified theories of Cognition*. Cambridge, Mass: Harvard University Press.
14. Nisbett, R. E., & Wilson, T. D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84, 3*, 231-257.
15. Ritter, F. E., (ed.) (2001). Special issue on using cognitive models to improve interface design. *International Journal of Human-Computer Studies*, 55, 1-14.
16. Spool, J., & Schroeder, W. (2001). Testing web sites: Five users is nowhere near enough. *CHI 2001 Extended Abstracts*, 285-286.
17. Vicente, K. J. (1999). Cognitive work analysis: Toward safe, productive, and healthy computer-based work. Mahwah, NJ: Lawrence Erlbaum Associates.
18. West, R. L., & Nagi, G. (2000). Situating GOMS Models Within Complex, Sociotechnical Systems. *Proceedings of Cognitive Science 2000*.