Modeling Performance Differences across Systems, Tasks, and Strategies

Jessica Lee (jessica.c.lee@nasa.gov)
San Jose State University, NASA Ames Research Center, Mail Stop 262-4
Moffett Field, CA 943035-1000 USA

Dorrit Billman (dorrit.billman@nasa.gov)
San Jose State University, NASA Ames Research Center, Mail Stop 262-4
Moffett Field, CA 943035-1000 USA

Abstract

Understanding problem-solving strategies and how different tools support problem solving is an important but difficult problem in cognitive science. Cognitive modeling provides one way of understanding and predicting problem solving and the impact of supporting software tools. Modeling typically requires tradeoffs between fidelity of result and difficulty of model building. We used CogTool to explore how well a limited modeling approach can predict performance differences between two applications that support problem solving, specifically, for planning attitude of the International Space Station. We develop a *modeling policy* for modeling complex behavior using a coarse-level tool with reduced expressive power; then we compare model predictions with experimental data to assess its ability to identify performance differences across systems, tasks, and strategies.

Keywords: problem-solving, strategies, HCI, modeling.

Introduction

Problem solving in the context of human-computer interaction both provides a resource for developing and testing cognitive models and generates complex situations of practical importance (Gray, 2008). The value of modeling problem solving outcomes and strategies in HCI is particularly high because empirical data may be impossible to collect at the point it would be most valuable. Specifically, a designer would like to know how design choices impact performance in advance of implementing a design. Thus, when performance data has its highest value, it can only be generated by model, not observation.

The need to predict performance has motivated development of several tools for HCI (Card, Moran, & Newell, 1980; John et al., 2004; Kieras, 2006; Patton & Gray, 2010). Most tools support model construction by providing a framework in which low-level component actions can be combined to represent larger problem solving tasks. Such tools can vary in the granularity of the low-level components it provides, in whether predictions are stochastic or deterministic, and in the complexity of tasks the tool can effectively model. Models also differ in whether the model generates alternative strategies (Smith et al., 2008) or more frequently, requires the modeler to specify the strategies to be modeled.

In selecting a modeling approach there is typically a tradeoff between the fidelity of the resulting model and the complexity of building it. Often, the cost of learning and constructing models is too high to justify the benefits of estimating performance times. CogTool (John et al., 2004; http://cogtool.hcii.cs.cmu.edu/) is an easy-to-use modeling tool that supports a simplified modeling process, while drawing on a well-vetted cognitive architecture, ACT-R (Anderson & Lebeire, 1998). The research reported here investigates how and how well a simplified modeling approach like that used by CogTool can predict performance times of complex problem-solving across systems, tasks, and strategies. We develop a method, our *modeling policy*, for modeling complex behavior using a coarse-level tool with reduced expressive power. We evaluate the strengths and weaknesses of this method by comparing model predictions with experimental data.

We first describe the work and tasks being modeled, planning by a NASA Mission Control group, Attitude Determination and Control (ADCO). Next we describe CogTool and why we selected it. We lay out the highlights of our modeling process, and describe a *modeling policy*, which we found helpful to consistently model a large and complex set of behaviors. We present results of comparing predicted and actual performance times. We conclude by discussing where and why modeling successes and failures occurred and what this suggests about using models to understand performance in complex HCI work.

ADCO Planning Domain & Software

ADCO controls the attitude (yaw, pitch, & roll) of the ISS (International Space Station). The operators monitor and command attitude in real-time and also develop plans in advance of real-time operations. ADCO plans specify the high-level activities (e.g. docking a Soyuz) and the actions (e.g. changes in control, maneuvering to a new attitude) that are needed to carry out the activity. ADCO currently uses legacy software (hereinafter called LEGACY; see Figure 1), which functions as a form-based text editor. Operators open a file for each activity and type in the parameters for each action within that activity. If an activity is rescheduled, the start and stop times of each action must be changed.

After analyzing needs (Billman et al., 2010), a new prototype planning application (hereinafter called NEW; see Figure 2) was designed. NEW provides better representations and operations, particularly for temporal relations. NEW allows rescheduling an activity as a whole, by sliding the activity in the timeline or by typing in new start times in the editing panel.

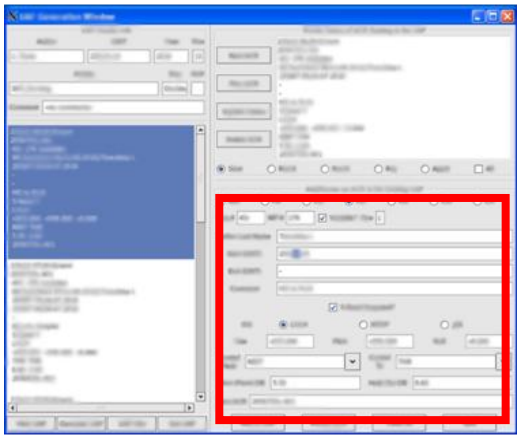

Figure 1. Screenshot of LEGACY system. Revision is done in lower right panel, by typing values into text boxes. (The attribute values shown do not reflect a real event.)

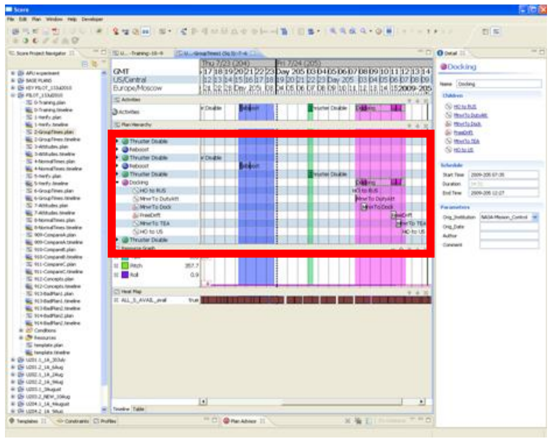

Figure 2. Screenshot of NEW system. Revision is done by dragging and dropping selected events on the timeline or by typing values in the panel on the right.

The experimental data to which we compared model predictions was a subset of an extensive experiment in which users performed a series of checking and editing tasks over two days, separated by one week. A between-subjects experiment compared performance using the two systems; 7 engineering students participated in LEGACY and 8 in NEW. We taught users about possible strategies to complete the tasks but left strategy choice open. In this paper, our experimental data draws from a particular set of editing tasks performed on the second day after 7-12 cumulative hours of practice on the system.

The particular set of interest consisted of 12 tasks, requiring users to shift the times of various events: 1) one action, 2) one activity, 3) a set of actions within one activity, or 4) a set of actions that span activities. The first two changes are common and the second two less so. We model performance on these time-shift tasks. Solving a “shift” problem requires the user to: 1) encode the problem; 2) select the event(s) to change, as one set or in subgroups depending on problem and strategy; 3) mentally compute the new start time; 4) set to this time. Steps 2-4 may iterate for subgroups. The user may check results or refer back to the problem description.

Modeling Environment: CogTool

Many tools have been developed to support modeling HCI tasks. One example is CogTool – a general purpose user interface prototyping tool that generates quantitative predictions of human performance, specifically response times of skilled performance. This paper is not an evaluation of CogTool, per se, but rather CogTool’s simplified approach to modeling behavior. We selected CogTool because it provided a good balance between required modeling effort and fidelity of result. It aims for simplicity by providing prepackaged interactive widgets (i.e. buttons, links) and transitions (i.e. click, hover) that connect between different states. It aims for validity by deriving the behavior of each component from ACT-R at the perceptual, cognitive, and motor level to predict time on task. The potential power of this modeling approach is that widgets and transitions can be linked together to generate predictions about complex sequences of behavior.

To build a model, a series of screenshots create a storyboard of frames. Each frame is overlaid with interactive widgets, and transitions representing user actions link frames to represent moving from one state to another. Each model represents a specific sequence of actions (such as a particular strategy on a particular task) that are demonstrated by clicking through successive frames. Based on the demonstration, CogTool constructs a Keystroke Level Model of how a skilled user would execute the task and it computes a predicted time. The resulting model built from demonstration can be modified by inserting additional components such as “look-at” or “think”.

While CogTool is capable of modeling unusual interfaces, its strength lies in prototyping standard widgets like menus, and buttons. For this experiment, constructing a model of LEGACY was very straightforward because the system utilized only standard widgets. However, we encountered many challenges in modeling NEW because it used complex interactions not directly supported by the CogTool library.

Modeling Policy

Our goal was to build a set of models that are individually accurate and collectively consistent, without excessive tailoring. Because we are trying to show adequacy of a limited modeling approach, demonstrating accuracy and consistency is difficult for two reasons. First, CogTool generates a separate model for every combination of task and strategy, thus requiring a large set of models to cover the behavior of interest. Second, the interactions in NEW cannot be modeled with CogTool widgets in a standard way. This leaves room for case-by-case variation in how to extend or apply CogTool. Consistency and low-tailorabilty are important and instrumental to validity. If individual models are tailored for each circumstance they are unlikely to generalize, to predict as well as post-dict, or to provide a valid model.

We formulated a modeling policy to help manage complexity, limit tailorability, and ensure consistency. The policy characterizes what situations fall within the scope of modeling and how models should be constructed scope.

Scope of Modeling

Users: Identifying what skill level the model represents is important for interpreting and applying its predictions. CogTool models the “skilled” user. To calibrate the models, each author performed blocks of simple tasks to generate various levels of skilled performance data. Comparing this data to CogTool’s predictions, the appropriate skill level emerges where users are making few errors and are familiar with the system, but tasks require deliberation and are not automatic. We found that performance of our participants on the last block of the second day aligned with skill level appropriate for CogTool.

Tasks: Understanding a model’s bounds in terms of task type and task complexity is important for deciding which tasks to model. With its library of vetted widgets and storyboard of discrete states, CogTool is best at modeling tasks using discrete “button pressing” actions, though it can be manipulated to represent continuous actions. The task complexity that CogTool can model is bounded by the granularity of available widgets and transitions. In selecting tasks to model, we started with simple discrete tasks and progressed toward continuous tasks of greater complexity.

Strategies: A strategy is an ordered set of actions to accomplish a task. Depending on the task, the set of strategies could range from a few to a very large number; thus, it is important to establish the scope of strategies to model. A CogTool model represents a single strategy for a single task as demonstrated by the modeler. Hence, CogTool can model simple tasks with little strategy variation, but any larger strategy space falls beyond CogTool’s scope of practical usage. Recognizing CogTool’s limitations in representing strategy for complex tasks, we chose to focus on strategies we observed participants using. Cogtool only directly predicts response time, not reasoning, decision making, or strategy selection. In many cases, users shift to efficient strategies with practice. Thus, speed of use may be a powerful predictor of strategy choice.

Environment: Characteristics of the system influence what types of interactions need to be modeled. CogTool is good at modeling interactions in discrete and stable interfaces. If users can change the display during interaction, modeling is more difficult; particularly, continuous change in the display is difficult for CogTool to model as it depends on a demonstration on a static display to capture and predict actions. We minimized this issue for CogTool by modeling selected items in which the display was not likely to vary across users or change within task.

Requirements on Model-Building

Our policy for model construction provides rules for breaking tasks down into component elements, and for how elements should be composed to model tasks and strategies.

The components for standard interactions such as button presses have widgets provided and can be modeled easily. For nonstandard interactions, modelers need to provide a fixed model component for the interaction type. Further, the modeler should construct models from existing components that are as similar as possible, to maximize consistency. Here are two examples of rules for components in nonstandard interactions:

Motor. The NEW system supports a drag-and-drop interaction in which the entity is dragged along a timeline and dropped at a specific time. Because the end location is very specific, this task requires fine motor control. Through the iterative modeling of experimenter-data on simple tasks, we developed a rule to model this end location as a very tiny widget; this was a modification made to the standard drag-and-drop model.

Perceptual. While most drag-and-drop interactions involve dragging an entity and dropping it at a visible target area, users in NEW have no visual cue for where to drop the entity. They instead rely on a separate, dynamically updated numeric display that indicates their progress toward the target. The standard drag-and-drop model was again modified to reflect this by inserting a “look-at” [time display] transition between the drag and the drop components.

After the components have been specified, they can be composed into sequences predicting more complicated behavior. Because CogTool components cannot be composed in parallel, it is important to select tasks that do not require parallel actions, or have overlapping actions that can be treated as sequential. In our case, even though drag-and-drop entails simultaneously moving the mouse while watching a target, the actions for this sequence could be reasonably stretched out and treated sequentially.

When modeling strategies for simple tasks, select strategies that are as general as feasible. This has the advantage that the strategy will be maximally reusable over task variations. We applied this policy to simplify models for typing in start times, by relying on an average strategy.

Process for Adhering to Policy

We used a structured method of incrementally extending and testing components. We verified the functionality of standard components in standard domains; we modeled new interactions by first testing single components in simple tasks and systematically incrementing the complexity of tasks, strategies, and components; we adjusted internal structure as needed. When comparing user data to model predictions, we prioritized the model’s ability to predict patterns of difficulty, not absolute times, because modeling individual differences in CogTool increases complexity and requires tailoring individual models. Adhering to our modeling policy was critical for ensuring the validity and consistency of model components and their composition. Further, this makes the resulting model set and its predictions easier to understand.

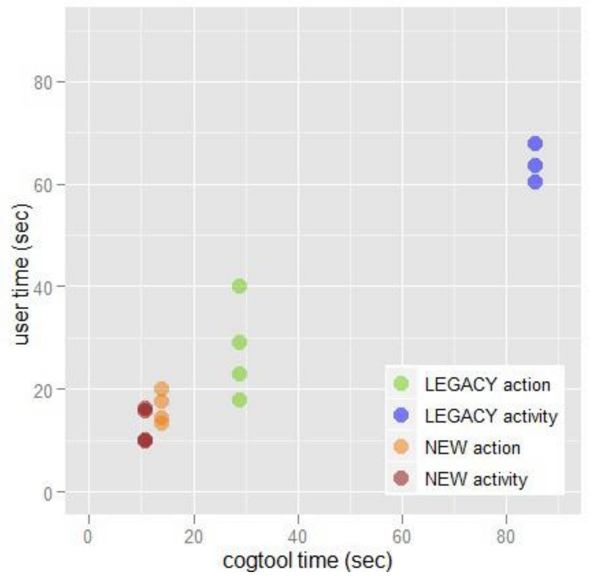

Figure 3. Times for users of NEW and LEGACY systems plotted against CogTool's predicted times.

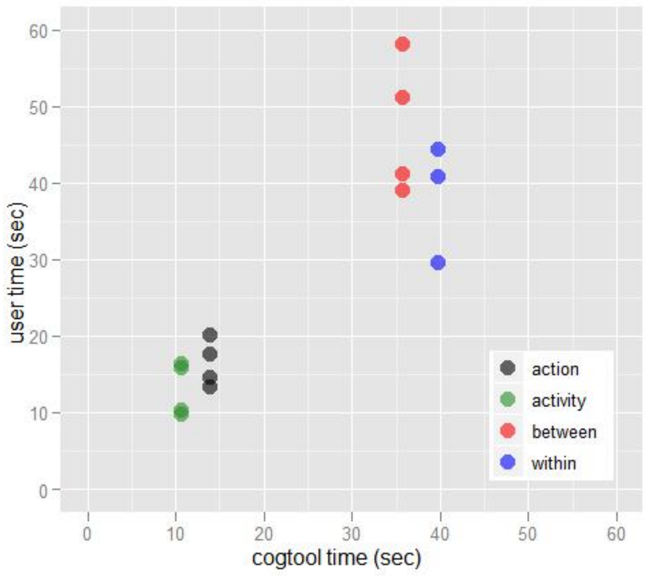

Figure 4. Times on four item types for users of NEW plotted against CogTool's predicted times.

Modeling Goals

Guided by the modeling policy, we constructed models at the system level, item level, and strategy level and compared them to experimental data. Generating models at these three levels of granularity provides insight into the practical value of modeling at different stages of the design process, and also provides a framework through which we can assess the strengths and weaknesses of the proposed modeling policy. Furthermore, how useful a model is depends on how well the model represents behavior. Having access to data from participants provides a way of assessing the validity of models.

Modeling System Differences

To compare the NEW and LEGACY systems, we modeled the two most common editing tasks – shifting an activity and shifting an action. We used data from skilled and errorless users, the four fastest participants on each system. (One outlying data point of 140s was dropped.) We created two CogTool models for NEW and two for LEGACY, modeling one activity shift item and one action shift item. The model for each condition used the most common strategy.

The predictions generated by the models were consistent with the user performance on NEW and LEGACY (Figure 3). NEW users (red & orange) were much faster than LEGACY users (blue & green) in both shifting activities and shifting actions and CogTool correctly predicted this. For NEW, activity shifts were slightly faster (users 13s (SE=1.8); CogTool 11s) than action shifts (users 16s (SE=1.5); CogTool 14s). For LEGACY, activity shifts were dramatically longer (users 83s (SE=18.7); CogTool 85s) than action shifts (users 28s (SE=4.8); CogTool 29s).

We were interested in CogTool's ability to post-dict overall performance difference between systems (though we had just four points to compare). Overall, there was a high correlation (r=0.999) between CogTool's predicted times and the experimental data in performing activity shifts and action shifts across systems. This affirms CogTool's ability to post-dict dramatic differences between two systems both on an absolute and relative scale.

Modeling Item Type Differences

To determine CogTool's ability to predict differences item types, we compared experimental data of four item types in NEW to their corresponding models. These four types were shifting actions, activities, actions within an activity, and actions across activities. We modeled NEW because it is both of practical interest and of greater complexity. We used data from the four fastest errorless users on NEW. (One outlying data point of 108s was dropped.)

We created CogTool models for each type of item. Each model used the strategy of the overall fastest user for the entire block; these strategies were commonly shared by other fast users. The fastest strategy for each task happened to be selecting the entities and editing the start times by typing in the details pane. In line with the modeling policy, we maintained consistency by modeling all time edits using the backspace key followed by typing in digits.

Average user performance was still highly correlated with the models' prediction times by item type (r=0.945) (see Figure 4). However, the order of difficulty was imperfectly predicted. For frequent items, CogTool correctly predicted that action shifts (users 13s (SE=1.8); CogTool 11s) would take longer than activity shifts (users 16s (SE=1.5); CogTool 14s). For the less typical items, CogTool's predictions were reversed (for actions: users 38s (SE=4.5); CogTool 40s versus for activities: users 47s (SE=4.4); CogTool 36s).

Despite the switched order for two of the item types, the values generated by the four CogTool models were broadly consistent with the experimental values for the four types of

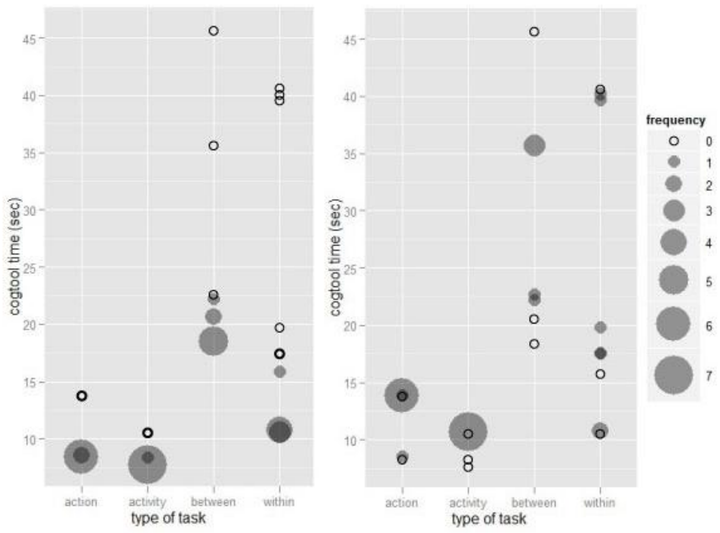

Figure 5. Left: Predicted choices: only fastest strategies chosen. Right: Actual strategy choices. Circle size shows number of users.

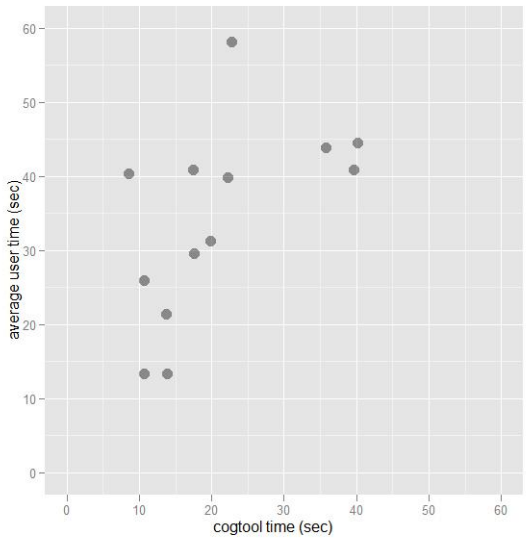

Figure 6. Average user times for strategy vs predicted time

items. This shows that CogTool can do an adequate job of predicting item type differences, especially for items that are structurally very different from each other. The order reversal for two tasks indicates possible weaknesses in modeling complex tasks. Two limitations of CogTool probably contribute to the failure to correctly predict the relative times of these two tasks. First, these tasks are more complex. As a result, there is greater variation in strategy even among skilled users, reducing the accuracy of modeling item difficulty with a single strategy. Second, CogTool models are purely mechanical and do not represent cognitive differences. In this case, shifting between activities is more cognitively taxing than shifting within an activity because there are more parts to keep track of. CogTool could be tailored, post-hoc by increasing "think" operators as needed, but this is inconsistent with our predictive modeling policy.

Modeling Strategy Differences

Turning to a finer granularity of modeling, we were interested in CogTool's modeling of strategy. First, we wanted to see if CogTool could predict strategy choice, from a collection of identified strategies. That is, is the strategy that CogTool predicts to be most efficient, the strategy preferred by fast, practiced users?

Second, we wanted to see how well CogTool could predict the actual times for those strategies. In order to compare the efficacy of various strategies, we used data from all eight NEW participants on each of the four items (representing the four item types). We removed data for responses with errors, outlying times, or other irregularity (such as redoing).

We then built models for each of these strategies using the Modeling Policy. Because CogTool cannot generate strategies, we created models post-hoc based on strategies chosen by users. For each type of item, we modeled every strategy used to complete the item plus a few additional strategies that we had (incorrectly) expected would be used. A total of 24 models were created, varying from 4 for the simplest task to 9 for the most complex. We also tallied the overall frequency with which strategies were used.

First, we wanted to see if CogTool could predict strategy selection. We took the strategy times generated by each CogTool model and ordered them from fastest to slowest for each item type. Because skilled users tend to shift toward faster times and CogTool can only predict time on task, we expected that if CogTool is a good predictor of strategy choice, most people would use strategies that CogTool rates as fast. For example, all users might select the fastest 2 or 3 strategies, as illustrated in the left panel of Figure 5. However, our findings showed that CogTool seldom predicted the use of strategies. For every item type, the strategies judged fastest by the CogTool model were not the ones commonly used (Figure 5, right panel). For the simplest items, the strategies that CogTool predicts are least efficient are the ones most chosen. For the more complex items, strategy choice is highly varied with little preference for strategies CogTool predicts to be fast.

Second, for those strategies that were used, how well did CogTool predict the time to use a given strategy? For each strategy used by any of the 8 NEW users, we found the average time and correlated the average data with the time predicted by the model. In comparing the times of strategies produced by users to analogous times predicted by CogTool, the correlation is modest (r=0.546), as shown by the wide scatter in Figure 6. (One dot may represent different numbers of users.) One key aspect of poor prediction is that all drag and drop strategies are lower than they should be.

Discussion

Summary of Results

We modeled performance on planning tasks by users working with two very different planning systems: one a legacy system currently in use by a Mission Control group and one a new system designed to match the work structure. We compared CogTool models to experimental data at the system, item, and strategy levels. The core strength of our approach was that by selecting CogTool and following our modeling policy, it was feasible to represent and get time predictions for a varied set of problem solving situations.

The models could predict the large performance differences between the very different systems, which provide different interfaces, interactions, and strategies. In addition, our models also provided reasonable correlations with data for different item types. It predicted simple differences between items that had clearly different interactions (activity vs action shifts), but was not as successful in predicting item differences for more complex items with overlapping strategies and characteristics. With respect to strategies, CogTool was not an adequate predictor of strategy choice selection nor did it do a good job of predicting strategy times.

Value of Validation

Detailed validation of a model in a complex work domain is difficult and rare (Gray, John, & Atwood, 1993). We selected a challenging work environment, which required interaction forms not previously supported in CogTool. For our model development, we vetted new components in simple models applied to one set of tasks and users (the authors), and applied this to a different, complex set of tasks and users (experiment subjects). Thus, we did not tailor the models to the data we sought to predict. We succeeded in accurately modeling behavior at a coarse but not fine level.

Successful prediction at a fine level would, indeed, be very useful from a practical perspective. It would be valuable to predict accurately and in advance what strategies are optimal, as we could then have taught these to users, to increase the likelihood that each system was being used to best advantage. With fine-grained accuracy, modeling could also be used to adjust design; for example, there are tradeoffs in design of the timeline layout between precision and scale; accurate models would allow exploring design alternatives to find best configurations.

Modeling Challenges

Our modeling policy highlighted several broad modeling challenges. 1) Behavior-based models (such as CogTool) have difficulty modeling working memory burden, presumably because this is least directly controlled by the task. Our problems differed in difficulty computing target times (add an hour vs 25 min), which affected component times and whether users included checking operations. Limited modeling of WM restricts the ability to distinguish between systems that impose different working memory burdens, a critical need for software supporting problem solving. 2) Assessing when a model of a component will compose cleanly in a larger model is difficult. Though our drag & drop models fared well on our simple test-bed tasks, when this component was included in larger models, these consistently underpredicted times. 3) Identifying when components will compose cleanly, without interaction, is critical. Problems with drag & drop models may have stemmed from interaction with other processes in the more complex models. While a richer modeling space can evaluate positive (e.g. parallel execution) and negative (e.g., competition for WM) interaction among components (Gray 2008; Smith et al, 2008), these models target simpler behaviors and hence entail greater complexity in building up to models of problem solving behavior. 4) Modeling human-computer, or human-automation, interaction requires a good model of the device. The default model of responsiveness and precision of mouse movements may have been inadequate.

Value of HCI Modeling

Modeling human-computer interaction provides both practical results and a test-bed for evaluating and developing modeling methods. Behavior here is constrained by the affordances of the interface, while still exhibiting a very rich range of problem solving phenomena.

Acknowledgements

Research was funded by the Human Research Program, Space Human Factors Engineering Project, NASA.

References

Anderson, J. R. & Lebiere, C. (1998). *The Atomic Components of Thought*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Billman, D., Arsintescu, L., Feary, M., Lee, J., Smith, A., & Tiwary, R. (In Press). Benefits of Matching Domain Structure for Planning Software: The Right Stuff. *Paper presented at the Proceedings of the ACM CHI Conference on Human Factors in Computing Systems.*

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

Gray, W. D. (2008). Cognitive modeling for cognitive engineering. In R. Sun (Ed.), *The Cambridge handbook of computational psychology*. New York: Cambridge University Press.

Gray, W., John, B., & Atwood, M. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. *Human Computer Interaction*, 8(3), 237-309.

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 455-462). Vienna, Austria: ACM.

Kieras, D. (2006). A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4. Retrieved January, 2011. from University of Michigan. Electrical Engineering and Computer Science Department FTP site: ftp://www.eecs.umich. edu/people/kieras/GOMS/NGOMSL_Guide. Pdf

Patton. E.W. & Gray, W.D. (2010). SANLab-CM: A tool for incorporating stochastic operations into activity network modeling. *Behavior Research Methods*, 42, 877-883.

Smith, M., Lewis, R., Howes, A., Chu, A., Green, C., & Vera, A. (2008). More than 8,192 ways to skin a cat: Modeling behavior in multidimensional strategy spaces. Paper presented at the *Proceedings of the 30th Annual Conference of the Cognitive Science Society*. Austin, TX: Cognitive Science Society.