Modeling Users at Multiple Levels of Expertise While They Learn

Jaehyon Paik (jaehyon.paik@lge.com)
UX LaboratoryLG Electronics, Seoul, South Korea

Jong W. Kim (jong.kim@ucf.edu)
Department of Psychology
University of Central Florida, Orlando, FL

Frank E. Ritter (frankritter@psu.edu)
College of Information Sciences and Technology
The Pennsylvania State University, University Park, PA

David Reitter (reitter@psu.edu)
College of Information Sciences and Technology
The Pennsylvania State University, University Park, PA

Corresponding Author

Frank E. Ritter, PhD

College of Information Sciences and Technology
The Pennsylvania State University
University Park, PA

Email: frank.ritter@psu.edu
Phone: +1 (814) 865-4453

24 May 2015

Document Statistics

Tables: 6 Figures: 7 Word Count: ~10,500 (inclusive of tables) (+abstract: 165; references ~1,500)

Abstract

We report a way to build a series of GOMS-like cognitive user models representing a range of performance at different stages of learning. We use a spreadsheet task across multiple sessions as an example task; it takes about 20~30 min. to perform. The models were created in ACT-R using a compiler. The novice model has 29 rules and 1,152 declarative memory task elements (chunks)—it learns to create procedural knowledge to perform the task. The expert model has 617 rules and 614 task chunks (that it does not use) and 538 command string chunks—it gets slightly faster through limited declarative learning of the command strings and some further production compilation; there are a range of intermediate models. These models were tested against aggregate and individual human learning data, confirming the models' predictions. This work suggests that user models can be created that learn like users while doing the task.

Keywords: User modeling, Learning, Expertise, ACT-R, GOMS, KLM.

1. Introduction

A model of how users learn has been acknowledged as one of the important goals in human-computer interaction (HCI) and in cognitive modeling. The GOMS[1]-type engineering model (Card, Moran, & Newell, 1983; John & Kieras, 1996b) has provided us with quantitative predictions of expert human behavioral performance. Since GOMS was introduced, the need has been recognized for a user model that learns as well (Olson & Olson, 1990). That is, GOMS models reflect only errorless skilled performance in terms of task time (although the Cognitive Complexity Method related to GOMSL has been used to predict learning and transfer on initial learning curves, Bovair, Kieras, & Polson, 1990). GOMS was a good start to look for a cognitive model of an expert behavior, but knowing how a model can represent users' learning behavior of a real world task has been and remains an intriguing question.

Cognitive architectures (e.g., ACT-R, Soar) provide important theories that scientifically summarize and model human behavior. People learn how to build a cognitive model by working with a cognitive architecture. However, it is acknowledged that people find it difficult to build cognitive models that represent complex human behavior (Pew & Mavor, 1998, 2007) and that it is difficult for cognitive modelers to learn details of cognitive architectures.

Using a mature cognitive architecture—i.e., ACT-R (Anderson et al., 2004) and Soar (Laird, 2012; Newell, 1990), cognitive modelers are able to model learning processes. We chose to use one of the widely used cognitive architectures, ACT-R, to

[1] Goals, Operators, Methods, and Selection rules

describe a larger model of learning in a real world task. (Other architectures could probably have been used as well.) The ACT-R theory, which provides a well-validated account for the acquisition of knowledge and its routinization, describes the process of human learning as the three-stage process through a distinctive classification of knowledge representation: declarative and procedural knowledge. In a cognitive model that is based on a production system (e.g., ACT-R), declarative knowledge is represented as a relational network of facts, and procedural knowledge is represented as a set of production rules. The applicability of each rule depends on the state of the system at a given point in time. Each rule at the symbolic level specifies *when* a cognitive act should take place (condition), as well as the effects of this act (action). Based on the construct of declarative and procedural knowledge, ACT-R provides both symbolic and sub-symbolic learning mechanisms—i.e., the production rule learning mechanism and the activation mechanism (Anderson, 1982b, 1987; Taatgen & Lee, 2003).

Despite the maturity of cognitive architectures, these efforts have been stymied in part by the detailed level of specification required by existing architectures to create models. While one of cognitive modeling's great strengths lies in providing computational features, the low-level abstractions have frequently proven expensive to create. Furthermore, it has been noted that these models have often proven difficult to maintain, extend, or merge (e.g., John, Blackmon, Polson, Fennell, & Teo, 2009; Pew & Mavor, 1998; Pew & Mavor, 2007; Ritter et al., 2003)

Another concern is that building a larger model easily is important because it, in fact, helps our understanding of cognitive functions and structures in the context of a real world task that relates to a daily work task (e.g., using a spreadsheet). However, a cognitive modeler can usually find it difficult to implement several hundreds of production rules, assuming that a modeler uses a production system to build a user model. A cognitive model often focuses on microscopic psychological tasks so that it can help examine cognitive functions in milliseconds, focusing mainly on the Cognitive and Rational Bands (Newell, 1990).

In this paper, we present a way to build a large model that learns and that learns differently in each learning stage—performance changes from a beginner through an intermediate to a skilled individual, and different mechanisms dominate each stage. We expanded a high-level behavior representation (Herbal) framework (Cohen, Ritter, & Haynes, 2010) to provide a general way to build larger cognitive models that learn. Herbal supports a general task analysis approach based on the Problem Space Computational Model (PSCM, Newell, Yost, Laird, Rosenbloom, & Altmann, 1991), a type of task analysis based on problem solving.

Here, we report analyzing a long (20~30 min.) non-iterated (sub-tasks are not repeated) task and building relatively large ACT-R models to predict performance from novice to expert. In addition, we tested the models against human data to test and validate them. In the next section, we briefly review previous efforts, and then discuss recent work on a high-level language for ACT-R and large learning models we have generated and tested, and then present a test of the models.

2. Review of Learning Models

In this section, we briefly review several types of cognitive models of learning used to model users. We also present the need for a high-level representation language that includes learning and interaction.

2.1 Static models without learning

GOMS-type frameworks have been widely used in HCI for designing user interfaces and predicting human behavior (e.g., Gray, John, & Atwood, 1993; John & Kieras, 1996a). GOMS models can predict the task procedure, task completion time, and inconsistencies in interfaces. GOMS has spawned several task analysis and modeling techniques, including the Keystroke-Level Model (KLM), GOMSL, and the Critical Path Method (CPM)-GOMS.

Despite the success of GOMS-type models, they have several limitations. Some user interface designers in HCI see GOMS as a relatively difficult technique to learn and use (John, Prevas, Salvucci, & Koedinger, 2004). There are several ways to make GOMS easier to use. CogTool (John, Prevas, Salvucci, & Koedinger, 2004) could solve this limitation by providing a graphical user interface (GUI). Users of CogTool can obtain the predictions of execution time easily, based on the KLM and the perceptual-motor component of ACT-R.

Although CogTool makes user predictions easier, it still has a limitation that GOMS has. That is, the GOMS-type models can only predict errorless skilled performance (although GOMSL could predict learning, CogTool does not include it). However, it would be useful to predict the time course of learning from novice through intermediate to expert. The GOMS framework and its extensions are a good start to predict expert behavior but knowing how a model can represent the process of learning a real world task would be a useful extension.

2.2 Cognitive architecture-based learning user models

Cognitive architectures, such as ACT-R and Soar, theories of the information processing mechanisms that are used in cognition, often provide learning mechanisms so that a cognitive model implemented under these architectures could predict learning during a task. Thus, GOMS-like models implemented in them could learn as well.

For example, ACT-R (Anderson, 2007; Anderson et al., 2004) is based on a production system; declarative knowledge is represented as a relational network of facts and procedural knowledge is represented as production rules. Each production rule has a condition/action statement and specifies when a cognitive act should take place. Based on the construct of declarative and procedural knowledge, ACT-R provides learning mechanisms and even a forgetting mechanism—i.e., the activation mechanism and the production rule learning mechanism (e.g., Anderson, 1982a; e.g., Anderson, 1982b; Anderson & Fincham, 1994; Anderson, Fincham, & Douglass, 1999; Pavlik & Anderson, 2005; Taatgen & Lee, 2003). More recently, ACT-R models were developed to match the results of brain imaging studies (Anderson, 2007), and its framework has a key role in some intelligent tutoring systems (Corbett & Koedinger, 1997).

Newell and Rosenbloom (1981) proposed an impasse-driven learning mechanism called chunking, which is the foundation learning mechanism in the Soar cognitive architecture. Soar has multiple learning mechanisms according to the different knowledge types: (a) it has chunking and reinforcement learning for the procedural learning, and (b) it has episodic and semantic learning for declarative learning (Laird, 2012). Chunking can happen when there is a lack of sufficient knowledge in a current problem space. Soar generates a subgoal to resolve the impasse. When this impasse is resolved or other results are produced, a new procedural memory is created. If Soar encounters a similar condition in the future, it can apply this newly learned chunk/rule to avoid the impasse. By adjusting numeric values that are relevant to rules, Soar includes reinforcement learning. Episodic learning are records of the contents of working memory while semantic learning and memory are related to storing and retrieving declarative facts.

Myer and Kieras (1997) developed a framework, EPIC, that can account for human information processing especially human perception, cognition, and motor activity for human computer interaction. With production rules, human performance (cognitive process) can be simulated and organized as a method to accomplish. Although EPIC does not have a learning mechanism in it, it has been widely used as a cognitive modeling technique and cognitive task analysis tools to explore human computer interaction.

Other cognitive architectures can be found in reviews (Langley, Laird, & Rogers, 2009; Morrison, 2003; Pew & Mavor, 1998; Ritter et al., 2003). Cognitive architectures have been widely used for modeling human behavior, understanding human cognition and problem-solving tasks. However, because those architectures use low-level languages, it can be difficult to create models using them. These features may lead cognitive modelers to concentrate on representing smaller and shorter tasks. We also argue that it would be better if we could generate several user models that predict a different range of expertise in a particular task with a single effort, similar to Card, Moran, and Newell's (1983) range of behavior. It would be much better if we can ease development of cognitive models through reuse (Langley, Laird, & Rogers, 2009; Ritter et al., 2003), however, the current range of cognitive architectures that we present above do not provide those features very strongly.

2.3 High-level languages for creating user models that learn

High-level behavior representation languages use abstractions to generalize common structures and processes found in existing cognitive architectures (Ritter et al., 2006). These persistent commonalities are evident when one considers defining a high-level knowledge representation, building a structured task analysis, or implementing a decision cycle characterized by the perceive-decide-act mechanism. Cognitive architectures' shared dependence upon *least commitment* (or the making of control decisions at every decision point) and *associative encoding* (or the associative retrieval of potential courses of action and a conflict resolution process for choosing between solution paths) entail a set of core commonalities from which to abstract. The commonalities include: declarative memory structures and retrieval methods, goals, procedural memory frequently used for the achievement of those goals, mechanisms for responding to external events, and an iterative decision process (Jones, Crossman, Lebiere, & Best, 2006; Langley, Laird, & Rogers, 2009).

These approaches can differ in many respects, such as their different representation structures, different reasoning and learning processes, etc. We will briefly summarize four existing candidate approaches for assisting to create more complex cognitive models: Jones et al.'s (2006) High Level Symbolic Representation Language (HLSR), Rosenbloom's (2009) graphical approach (Sigma), Reitter and Lebiere's ACT-UP, and Herbal, a High-Level Behavior Representation Language (Cohen et al., 2010).

HLSR uses three primitives (relations, transforms, and activation tables) to create micro-theories for representing cognitive models or architectures (and by extension, cognitive theories). It has two approaches; (a) a top-down approach by analyzing similarities across a numerous cognitive architectures and (b) a bottom-up approach by providing a common language that generate both ACT-R and Soar models. Those approaches enable modelers to develop cognitive models more easily.

Rosenbloom (2009) seeks to develop a unified implementation level based upon factor graphs. According to Rosenbloom, a cognitive architecture can be represented using graphs, because it has a fixed structure for the human mind, and there are relations of knowledge and skills that are embodied within the architecture.

ACT-UP is a high-level implementation of ACT-R that emphasizes rapid modeling and reusability. It asks the modeler to specify algorithms in a functional programming language rather than in procedural rules, committing only to verifiable portions of the model, and underspecifying unclear strategies (Reitter & Lebiere, 2010). The authors report writing a complex model (Reitter, Keller, & Moore, 2011) in 10% of the lines of code in the ACT-R variant in a fraction of the time.

Herbal characterizes common cognitive modeling tasks such as task analyses and problem solving using an ontology based upon the Problem Space Computational Model (Newell, Yost, Laird, Rosenbloom, & Altmann, 1991). Its ontology consists of agents, problem spaces, conditions, actions, and types (Cohen, Ritter, & Haynes, 2010, also see the Herbal web site, tutorial, and sample models). Each of these approaches is promising; each potentially allows for comparative analysis across architectures. Each, if fully developed, could promote model reuse across a diverse community of users.

Herbal, however, has some unique features among the four approaches. HLSR supports both Soar and ACT-R, but is not yet available outside of its developers. The results of an unpublished usability study (*N* = 23 in three between-subjects conditions) comparing Soar, ACT-R, and HLSR show that there are few reliable differences between these approaches in several tests, except some tasks might be coded about 2x faster with HLSR[2]. Sigma is still at an early stage and remains focused on re-implementing aspects of Soar on a more functional level.

Herbal, in contrast, is open source, supports two cognitive architectures and one agent architecture across a set of common cognitive modeling tasks (Soar, ACT-R, and Jess), has been tested with several usability studies (e.g., Cohen, 2008; Cohen, Ritter & Haynes, 2010, 2012) to improve it and document its effect on coding time, has been used to create several models (Cohen, Ritter, & Haynes, 2007; Cohen et al., 2010; Friedrich, 2008; Paik et al., 2010), and has been used for a docking study, which compares how two similar models perform, and considers what would be needed to 'dock' them to produce the same results (Burton, 1998). The docking study was done by developing a previously existing model (Pirolli, 2007) in ACT-R and comparing its results to a Herbal-created model (Zhao, Paik, Morgan, & Ritter, 2010). So, we have chosen to develop our model using Herbal.

[2] We thank Jacob Crossman for providing us with a copy for our use.

2.4 Summary

In this section, we briefly reviewed the currently used cognitive modeling approaches that allow cognitive modelers to develop models of users more easily, starting with GOMS. Several cognitive architectures, such as ACT-R and Soar, have been used to create models because of the limitations of GOMS-type engineering models, such as the difficulty of use and inability for representing novice users and their learning process. It is difficult to make cognitive models using these cognitive architectures, because they use low-level languages. This difficulty may have led cognitive modelers to concentrate on representing smaller and shorter psychological tasks.

The reimplementation of cognitive modeling languages using object-oriented languages (e.g., Java) may be useful for programmers who are familiar with those languages, but does not address the fundamental problem of choosing an abstraction level that is unsuitable for complex tasks and partially unknown strategies.

The high-level cognitive languages that are reviewed could help resolve these problems. Among them, Herbal has several advantages for user modeling. It has undergone several usability tests, has been used to created several models, has had a docking study, and it is published under an open source license. Next, we will describe Herbal and work related to Herbal more fully, focusing on Herbal's implications for HCI and the more rapid creation of user models that can learn.

3. Herbal

In this section we provide a description of Herbal. We also present the recently added ACT-R declarative memory pane compiler (Herbal/ACT-R compiler), which uses the key components of Herbal and was used to build a range of ACT-R models in a spreadsheet task.

3.1 Overview of Herbal

Herbal's ontological representation is based on the PSCM. It defines behavior as operators modifying states, as well as change through choosing and using problem spaces. The elaboration cycle describes the process by which an agent modifies its state representation through the associative retrieval of information. The agent achieves this through the firing of production rules (conditions and actions): conditions are the circumstances under which that information is relevant; actions specify the knowledge to perform or apply. The decision cycle in turn consists of repeated cycles of elaboration that persist until quiescence—until no further rules can be fired.

The agent makes decisions based upon its state interpretation and preferences, choosing either a unique operator (actions capable of transforming the state) or generating an impasse if an operator cannot be selected due to insufficient knowledge. Agents resolve impasses by generating sub-states that enable the agent to retrieve the information necessary to specify the next operator. Problem spaces are thus representations describing a sequence of decisions (or a search in the event of limited knowledge) that can be further defined in terms of goals, states, and operators.

Herbal's ontology characterizes behavior in terms of classes that represent concepts such as states, operators, elaborations, impasses, conditions, actions, and working memory. These classes furthermore entail basic relationships for instance—states can contain impasses, working memory, operators, elaborations, and other information while operators and elaborations can contain zero or more conditions and actions. Programming in Herbal thus involves instantiating objects using these ontological classes. Herbal also supplies additional attributes that enable future developers to understand the model including its design rationale, the intent motivating creation of a given object. Including the design rationale can lead to models that can explain themselves (Haynes, Cohen, & Ritter, 2009).

Users create models by editing Herbal's classes either graphically with an Eclipse plug-in or directly in XML. The plug-in affords users access to Eclipse's functions to assist them in creating and maintaining models in a graphical interface. While Eclipse can simplify the creation of PSCM components, some developers prefer to work directly with the Herbal high-level language in XML. Developers can edit the Herbal XML code directly, and these changes are immediately reflected in the GUI Editor (Friedrich, Cohen, & Ritter, 2007). Herbal then compiles the XML representation into low-level rule-based representations that are executed within a lower level architecture, Soar, or Jess, and, as we report here, ACT-R. More information on Herbal is available from its web site (acs.ist.psu.edu/herbal) including a manual, and sample models.

Herbal clarifies for the user the model's structures and relationships, and makes the high level structures (e.g., operators, problem spaces) available as first class structures. In general, the arguments for using a high level behavior representation language for modeling are similar to all arguments for high level languages (Brooks, 1975), including ease of use, programming speed and uniformity, and intelligibility of the resulting model/program (Cohen et al., 2010; Dancy & Abuomar, 2012; Ritter et al., 2006). In the following two sections, we briefly describe applications and usability tests of Herbal before discussing recent work extending Herbal to include an ACT-R compiler.

3.2 Applications of Herbal

Herbal has been used to create several models that learn. To provide some context, we describe briefly four representative models, as well as work using Herbal to develop an intelligent user interface.

Cohen et al. (2007) tested a model of learning and unlearning by implementing a competitive reflective learning model and also opponents in Herbal/Soar. Participants (n = 10) and a learning model played a simplified baseball game, acting as the model pitchers attempting to strike out batters. Each participant faced batters employing one of five batting strategies: *hacker* (always swinging), *aggressive* (always swinging at the first pitch and when there are fewer strikes than balls, unless there are three balls and two strikes), *random* (randomly swings), *chicken* (never swings), and *alternate* (swings if the last pitch was a fast ball and does not swing at the first pitch or if the last pitch was a curve ball). The pitcher model learns to pitch and the batter models (written in an afternoon) respond to the model or human participant by cycling through all five batting strategies (presented in blocks). Participants and the model continued playing the game until they had struck out seven batters in a row. The model's learning compared well to humans doing the same task.

Herbal has also been used to create a revised version (Friedrich, 2008; Friedrich & Ritter, 2009) of the Diag model (Ritter & Bibby, 2008). The original Diag model, written in Soar, predicted fairly accurately (r^2 > .95) the time course of learning and problem solving in a troubleshooting task for 8 out of 10 subjects. The data Friedrich gathered was done in a way to lead to more strategies. Herbal was used to create five more strategies and was used to find further combined strategies (18 out of 37 had significant r^2's and 25 were greater than .5) in the additional subjects' behavior.

Herbal has also been used to create a model of an anti-terrorism force protection planner as part of the Rampart project (Haynes, Kannampallil, Cohen, Soares, & Ritter, 2008). This model is embedded in a complex decision-support environment that assists users in selecting between various resource allocation options.

A version of Pirolli's (2007) hotel price-finding ACT-R model created using Herbal was compared (docked) to a similar model created by hand, and we found that the models match each other's performance. Furthermore, Herbal has been used to create a Soar model in an adversarial reasoning task in NSS (Dancy & Abuomar, 2012).

These projects provide some illustration of Herbal's versatility. In each case, we could not have developed models of equivalent complexity or interest as quickly in the underlying low-level formalisms. These projects suggested two things: one, that it would be useful to have a compiler in Herbal for ACT-R (e.g., Anderson, personal communication, has asked how would an ACT-R model of the Diag task perform?); and two, it would be useful to have a compiler that supported representing task analyses more directly. We report a recent extension of Herbal that provides these two features, and an implemented model that begins to test them showing how learning from novice can be modeled.

3.3 *Tests of Herbal's Usability*

We have tested how much faster Herbal is to create models several times. A summary is shown in Table 1. The table uses a baseline rate of 3.6 min/production. This rate was reported by Yost (1992, Table 4-2, 4-3, and Yost, 1993, finding 3.6 min./production as the median across tasks in three experiments) as a time to write Soar production rules using his TAQL high level language. Yost's three subjects were graduate students in computer science at Carnegie Mellon University (and included himself). They created a variety of relatively small AI and logic tasks. Yost does not report times for Soar, but an anecdotal rule of thumb that was used at the time was 5 min. per production.

Table 1. Tests of Herbal's usability.

Publication	Speed up	Time per prod. (min.)	Population	N	Task
Morgan et al. (2005)	16%	3	PSU under-graduate (UG)	1	Tank game
Cohen et al. (2007)	n/a	n/a	PSU graduate student	1	Baseball pitching
Cohen (2008)	44%	2	Lock Haven Psych, CS & CIS UGs	24	Vacuum cleaner simulation tasks

In Table 1, Morgan et al. (2005) reported a study of a single Penn State undergraduate writing a single tank model to play a tank game. Cohen (2007) reported creating 6 models with about 10 rules per model in an afternoon, but did not report the model creation rate because he did not think it would be believed (Cohen, personal communication, 2007). In his thesis, Cohen (2008) explored teamwork and maintenance using Herbal. He had teams of undergraduate students build models. One student would build components, a library, and then another would use the components to build a model to move in the Vacuum cleaner world (Cohen, 2005, Table 7-4). The students working serially created 16 production models much faster than Yost reported, but perhaps more importantly, half of these students were undergraduate psychology majors at a teaching college, and their performance was indistinguishable from undergraduate CS majors there, and faster than CMU graduate students.

Herbal has also been used to create larger models where timing was not recorded. Friedrich (2008) used Herbal to create six models of multiple strategies on a troubleshooting task. The previous project created one model of this task. Friedrich did not report time per production, but created 6 times more models than previous work.

There are several limitations to these data. The tasks are not as big as some expert system tasks and some models, and the models are not hardened to be systems, a criterion that Brooks (1975) notes makes systems slower to be developed. The subjects are probably not representative of commercial developers who both have more experience and also more distractions. Interaction with the task remains a problem as well. The models created here had tasks immediately available to them to interact with—supporting models' interactions with tasks can be problematic and take time to create and debug (Ritter, Baxter, Jones, & Young, 2000). Overall, however, these results suggest that Herbal/Soar allows less experienced modelers with less background (i.e., psychology vs. computer science) to create models more quickly.

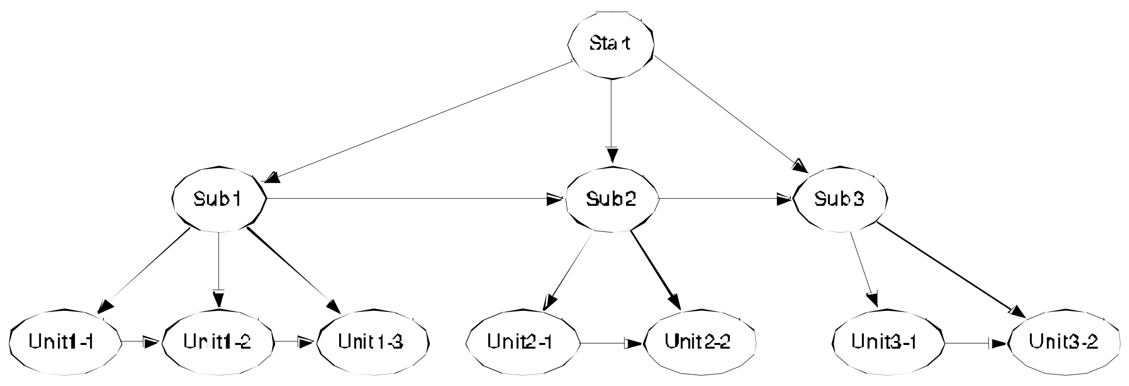
3.4 The Herbal/ACT-R Compiler

We have created a task-analysis-based compiler in Herbal to create ACT-R models that perform a hierarchical or sequential task. The compiler is based on representing task knowledge in a hierarchical task analysis in declarative memory. With this approach, we were able to add hierarchical and sequential tasks to an ACT-R model —the relations among tasks are shown in a tree form in the user interface. Herbal then makes declarative memories and production rules based on these relationships, and based on those chunks and rules the Herbal/ACT-R compiler can generate either a novice ACT-R model or eleven kinds of expert ACT-R models with varying degrees of expertise ranging from 0% to 100%. In this case, expertise is based on the task completion times, that the times are relatively lower than novices require to perform the task, and similar to how these terms have been used previously in HCI (Bovair, Kieras, & Polson, 1990; Card, Moran, & Newell, 1983).

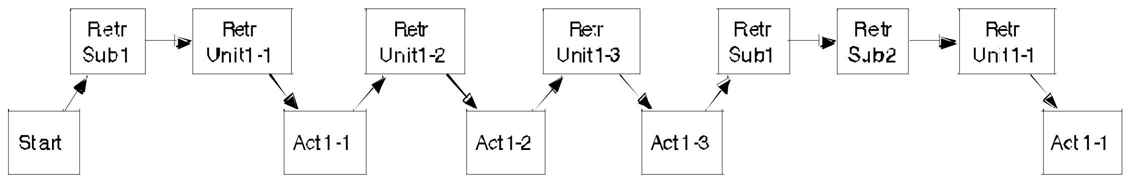
Novice models, in this framework, have no information regarding the next task step, and thus must retrieve each step from declarative memory, whereas the expert models have the next task step incorporated as part of the operation some proportion of the time. The proportion of the time the step is known is used to label the model (e.g., 10% of the time is the 10% expert). The novice model thus predicts the maximum anticipated completion time while the *normative expert* models (described below) provide the task time for a range of expertise including complete experts.

Distinguishing novice from expert, we further divided the expert models into two types: (a) a *normative expert*, model where all the declarative memory elements for the task has been compiled into procedural knowledge, and (b) *practicing experts*, models that exhibit varying degrees of proceduralization. The model exhibiting 100% expertise (*normative expert*) provides a baseline. It does not use memory elements in declarative memory to perform a task because the model has these elements fully proceduralized. Models ranging between 0% and 90% expertise (*practicing experts*) have a proceduralized task structure, but the number of declarative memory retrievals to walk the task structure varies. For example, if a model needs to have ten declarative memories (DMs) to perform a task, the 0% expertise model needs to retrieve all DMs to perform a task while the 10% expertise model needs nine DM retrievals, because one (10%) declarative memory is already proceduralized. The rest of the models, such as 20%, 30%, etc, perform a proportional amount of declarative memory retrievals with this mechanism. (The 0% expert model differs from the novice model in that the 0% expert one knows which memories to retrieve, however, the novice one has to fully walk the task tree in declarative memory to find the steps to do.) The *practicing expert* models thus provide us with a basis for making useful comparisons with the human data by providing incremental predictions of performance (task completion time) based upon expertise, and will perhaps enable us to isolate the participants' actual average level of expertise at the onset of learning.

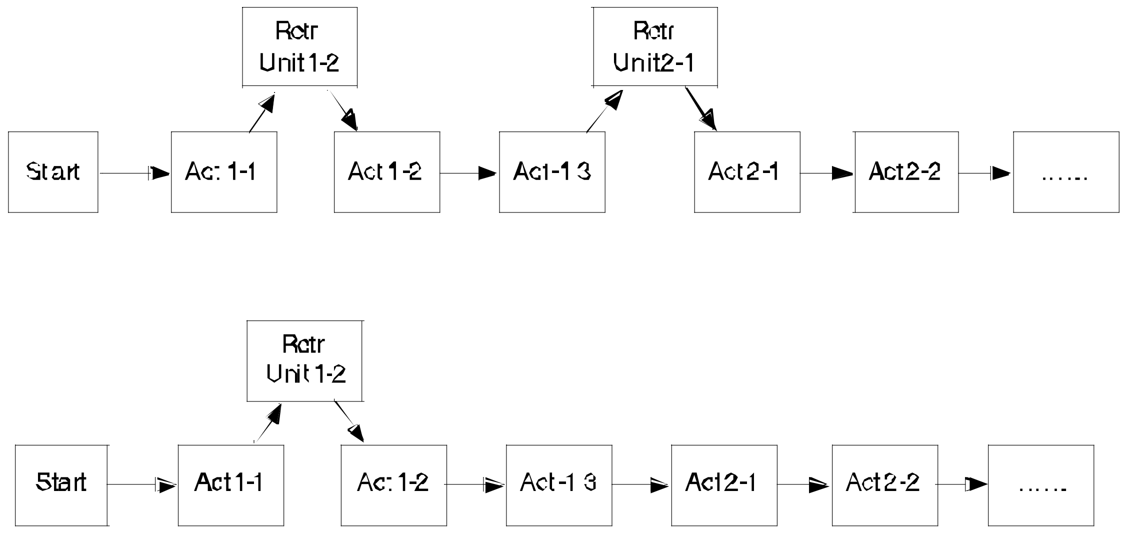
Figure 1 shows (a) the declarative memory structure of the two types of models, and (b, c) examples of the mechanism to perform a task with respect to different levels of expertise. As can be seen in Figure 1, the novice model walks through all the nodes of the task structure based on a depth-first search method to perform a task. However, the practicing expert models need to retrieve only a proportional amount of information based on each models' expertise.



(a) An example of the model's declarative memory structure, with sub-tasks 1-3 shown and unit tasks below them. Each node represents a chunk and each arrow represents the hierarchical relationship between chunks.



(b) An example trace of the mechanism to perform a task in the Novice Model.



(c) Example traces of the chunks used to perform a task in the practicing expert models.

Figure 1. The declarative memory structures for the models (a), and the example traces of the mechanisms to perform a task with respect to different levels of expertise (b, c).

4. Description of the Empirical Data

Here, we compare the learning performance data (subjects using a menu-driven interface) with our models' performances. We used published human performance data on the Dismal spreadsheet task (Kim & Ritter, 2015). The Dismal spreadsheet task is sequential, consisting of 14 subtasks; tasks are performed in order once per session and not repeated multiple times, such as in Argus (Schoelles & Gray, 2001) and similar tasks where a single task is performed through the duration of a session. Table 2 shows the subtasks of the Dismal spreadsheet task.

Table 2. The fourteen subtasks in the Dismal spreadsheet task.

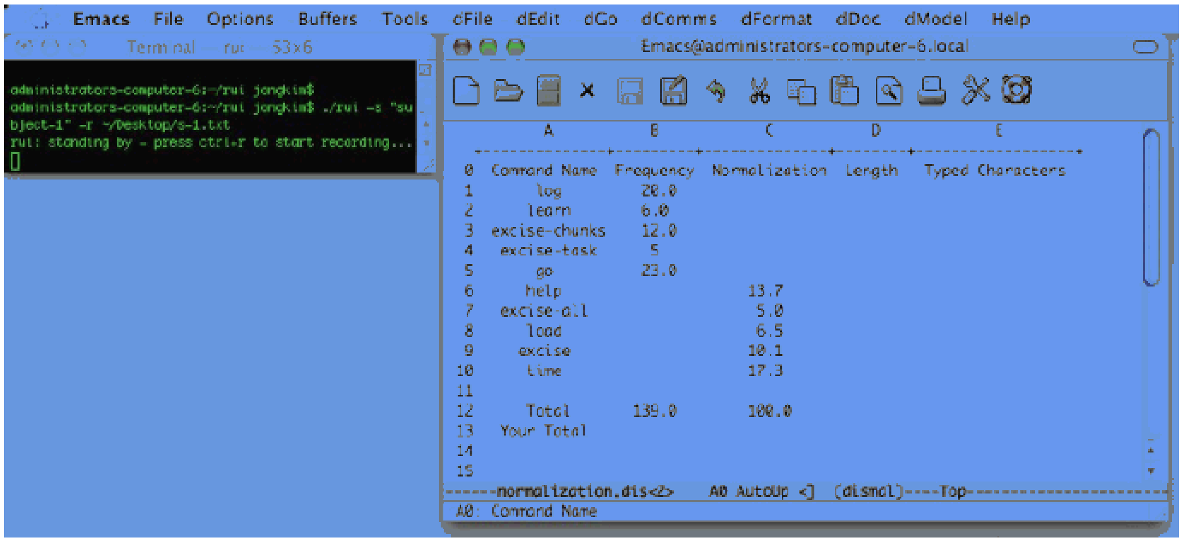
(1)	Open a file, named *normalization.dis* under the *experiment* folder
(2)	Save as the file with your initials
(3)	Calculate and fill in the *Frequency* column (B6 to B10)
(4)	Calculate the total frequency in B13
(5)	Calculate and fill in the *Normalization* column (C1 to C5)
(6)	Calculate the total normalization in C13
(7)	Calculate the *Length* column (D1 to D10)
(8)	Calculate the total of the *Length* column in D13
(9)	Calculate the *Typed Characters* column (E1 to E10)
(10)	Calculate the total of the *Typed Characters* column in E13
(11)	Insert two rows at cell A0
(12)	Type in your name in A0
(13)	Fill in the current date in A1 using the command *dis-insert-date*
(14)	Save your work as a printable format

4.1 Method

4.1.1 Materials

Subjects performed the Dismal spreadsheet task using a vertical mouse (Evoluent Vertical Mouse), a Macintosh keyboard, an Apple desktop computer, and a 20" display. The task completion time and keystrokes, mouse clicks (pressed and released), and mouse movements (e.g., *xy* coordinates of mouse locations in pixels) were recorded by the Recording User Input (RUI) system (Kim & Ritter, 2007; Kukreja, Stevenson, & Ritter, 2006; Morgan, Cheng, Pike, & Ritter, 2013).

Figure 2 shows the study environment with RUI and Dismal. In the Dismal spreadsheet, some default values were provided in the Frequency and Normalization columns. Thus, participants worked on the same spreadsheet problems across sessions, but the data given were varied.


Emacs File Options Buffers Tools dFile dEdit dGo dComms dFormat dDoc dModel Help
Terminal — rui — 53x6
Emacs@administrators-computer-6.local

```
administrators-computer-6:~/rui jongkim$
administrators-computer-6:~/rui jongkim$ ./rui -s "su
bject-1" -r ~/Desktop/s-1.txt
rui: standing by - press ctrl-r to start recording...
```

```
          A            B             C            D          E
   ----------------+---------+-------------+--------+------------------+
0  Command Name  Frequency  Normalization  Length   Typed Characters
1      log         20.0
2     learn        6.0
3  excise-chunks   12.0
4   excise-task     5
5      go          23.0
6     help                      13.7
7   excise-all                   5.0
8     load                       6.5
9    excise                     10.1
10    time                      17.3
11
12    Total       139.0        100.0
13  Your Total
14
15
------normalization.dis<2>    A0 AutoUp <]  (dismal)----Top----------------------
A0: Command Name
```

Figure 2. The Dismal spreadsheet task study environment. RUI is seen on the left and ready for recording user performance. Dismal on the right is ready for user input.

4.1.2 Participants

A total of 78 students at Pennsylvania State University took part in the original experiment for compensation (Kim & Ritter, 2015). The first 6 participants were pilot subjects and 12 participants could not complete the multiple experiment sessions due to personal time conflicts (e.g., a job interview that arose after starting the study). Thus, a total of 60 completed all of the experimental sessions; 30 subjects (randomly assigned) using the menu-driven interface and 30 subjects using the keystroke-driven interface. No participants had previous experience with the Dismal spreadsheet, Emacs, or this task. Menu-driven users all reported no experience with the vertical mouse. All participants that completed all required sessions were paid in full. There were no incentives based on performance.

4.1.3 Design

The experiment consisted of two independent factors of input modality (keystroke-driven modality and menu-driven modality). In this paper, we are only interested in modeling the mouse-driven modality.

4.1.4 Procedure

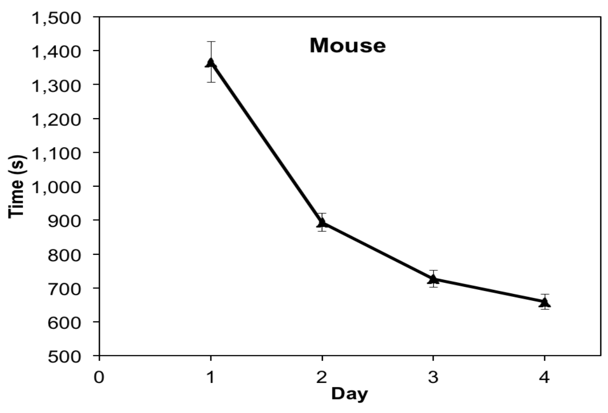
For the learning sessions, all subjects completed a series of study sessions for four consecutive days, Day 1 to Day 4. In the study sessions, subjects used the study booklet to learn the Dismal spreadsheet task knowledge (this use was not recorded). The duration of each study session was no longer than 30 minutes. After subjects studied the booklet, they performed the Dismal spreadsheet task. While doing the task they had access to the study booklet. For example, on Day 1, participants had a maximum of 30 minutes to study the given spreadsheet task and then performed the Dismal spreadsheet task. On Days 2 to 4, subjects were allowed to refresh their acquired knowledge from Day 1, using the study booklet, and then performed the task.

4.2 Results

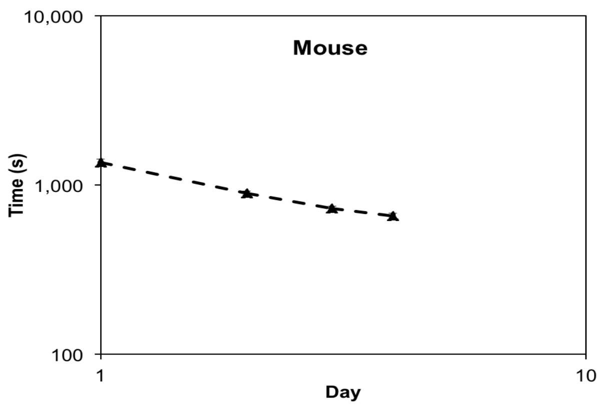
All of the menu-driven interface thirty-subjects completed the learning sessions, which includes performing the subtasks in order. Their mistakes were not analyzed formally, but they were not catastrophic (i.e., they managed to complete the task in all sessions). Mistake correction times are included in the task completion times. The average task completion time ranged from 1,366 s (*SE* = 60.8 s) on day 1 to 659 s (*SE* = 22.7 s) on Day 4, shown in Table 3. Figure 3 shows the mean task completion time in a linear and log-log scale (Newell & , 1981). The learning data matches the power law of learning (y=1,338 $x^{-0.5}$, r^2 = 0.99, for the menu-driven modality group, fit in log-log space).

Table 3. *Task completion times in seconds for the four learning sessions.*

	Day 1	Day 2	Day 3	Day 4
M	1,366	894	727	659
SE	60.8	26.6	25.5	22.7



(a) Task completion times for four learning sessions. Error bars show *SEM*.



(b) The log-log plot of learning data. Error bars show *SEM*.

Figure 3. Learning performance of the menu-driven modality group with error bars showing *SEM*.

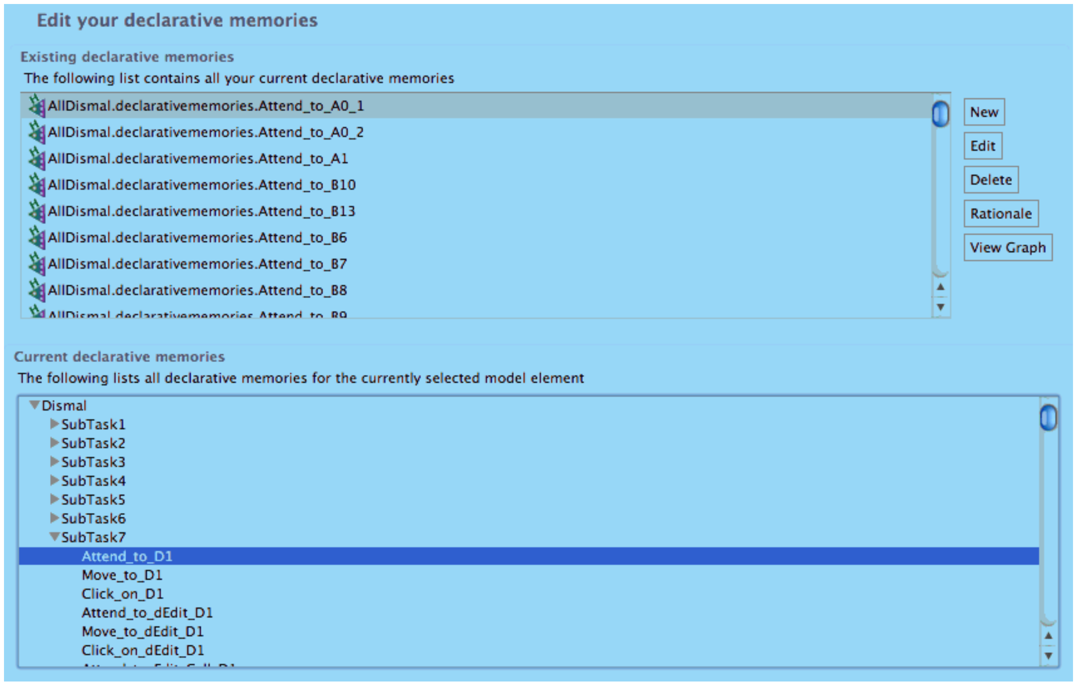
5. Description of Model Predictions

There are two ways in Herbal to make an ACT-R model for the Dismal spreadsheet task[3]. One is using the Herbal declarative memory pane (GUI), and the other is using the corresponding memory elements in the XML representation. We used the XML representation to enter the information of each unit task and their relationship, because this task has numerous declarative memory elements and unit tasks in each subtask, and these tasks are repeated in a very similar sequence.

After entering the whole task, the Herbal/ACT-R compiler translated this XML file into Herbal as a tree structure. Figure 4 shows the XML structure of the declarative memory elements and the structure of the Dismal spreadsheet task in Herbal.

[3] The models can be downloaded at http://acs.ist.psu.edu/paik/Dismal_Models.zip

```
<xs:element name="declarativememories">
      <xs:element name="declarativememory">
             <xs:element name="parents"> </xs:element>
             <xs:element name="firstchild"> </xs:element>
             <xs:element name="nextsibling"> </xs:element>
             <xs:element name="action"> </xs:element>
             <xs:element name="perceptualmotor"> </xs:element>
             <xs:element name="chunktype"> </xs:element>
             <xs:element name="key"> </xs:element>
             <xs:element name="nextperceptualmotor"> </xs:element>
             <xs:element name="prerequest"> </xs:element>
      </xs:element>
</xs:element>
```


Edit your declarative memories
Existing declarative memories
The following list contains all your current declarative memories
New
Edit
Delete
Rationale
View Graph
Current declarative memories
The following lists all declarative memories for the currently selected model element

AllDismal.declarativememories.Attend_to_A0_1
AllDismal.declarativememories.Attend_to_A0_2
AllDismal.declarativememories.Attend_to_A1
AllDismal.declarativememories.Attend_to_B10
AllDismal.declarativememories.Attend_to_B13
AllDismal.declarativememories.Attend_to_B6
AllDismal.declarativememories.Attend_to_B7
AllDismal.declarativememories.Attend_to_B8

Dismal
SubTask1
SubTask2
SubTask3
SubTask4
SubTask5
SubTask6
SubTask7
Attend_to_D1
Move_to_D1
Click_on_D1
Attend_to_dEdit_D1
Move_to_dEdit_D1
Click_on_dEdit_D1

Figure 4. The XML structure of the declarative memory elements and the structure of the Dismal spreadsheet task in the Herbal GUI.

5.1 Novice model

The novice model has 1,152 declarative memory elements (614 for the process of the task, and 538 for the characters to be entered) representing the task and 29 production rules. Each chunk of declarative memory about the task has a parent, first child subtask, next sibling task, and slots that contain perceptual-motor tasks to be performed. Using the relationship among the tasks, Herbal generates in ACT-R a Dismal spreadsheet model hierarchically as a specific sequence of tasks. Among the 29 production rules, 19 rules walk through the hierarchical tree structure, and retrieve the next node from the declarative memory elements according to a depth-first search algorithm to complete the task. The other 10 rules are applied when the model uses the perceptual-motor components of the ACT-R architecture.

5.2 Expert model

The expert model with 100% expertise does not use any of the declarative memory elements describing the task steps to perform the task. It only uses 538 declarative memory elements for recalling strings to type in the task. (The model already knows the whole process of the task and the sequence of each subtask and unit task in a proceduralized way.) The declarative memory elements for the process of the task could, of course, be included in the model, but would not be used by this model. The 617 production rules follow the number of unit tasks in the Dismal spreadsheet task and the typing task, so the model follows these steps according to the sequence of the whole task.

5.3 Levels of expertise models

We have 10 different models between the novice and the expert. Each model has a different level of expertise, from 0% to 90%. We represent the expertise using different numbers of declarative memory chunks and the extent that declarative memory chunks are retrieved by production rules. Each declarative memory element contains the next task information, so the model can walk through the whole task. The model with 0% expertise has 1,152 declarative memory elements, the same as the novice model, and it has 617 production rules. This model always retrieves declarative memory chunks to move to the next step, but it knows which chunks to retrieve, and does not have to follow a tree to find the next action as the novice model does.

The difference between the 0% expertise model and novice model is that the 0% expertise model has the exact information of the next step to do (the next declarative memory to retrieve), however, the novice model has to carry out more memory retrievals to walk through the hierarchical tree structure using a depth-first search algorithm.

As expertise increases, the number of declarative memory elements used by the model decreases. This is because the meaning of expertise in a task is the amount of knowledge, information, and experience of that task already proceduralized, so the model does not need to retrieve the next steps from declarative memory. For example, the 10% expertise model uses 1,091 (614*0.9 +538) declarative memory elements (chunks), the 30% expertise model uses 968 (614*0.7 + 538) chunks, the 50% model uses 845 (614*0.5 + 538) chunks, and so on. Table 4 shows the distribution of the number of declarative memory elements and production rules for the models.

These 12 models (one novice, one normative expert, and 10 practicing expert models) use the ACT-R perceptual-motor modules to interact with the task simulation.

Table 4. The distribution of the number of declarative memory elements and production rules for models.

		Models with Expertise					
	Novice	**0%**	**10%**	**…**	**50%**	**…**	**100%**
Decl. mem. elements used	1,152	1,152	1,091	…	845	…	538
Production rules	29	617					

5.4 The models over multiple runs

Running these 12 models, we confirmed that the novice, intermediate, and expert models perform the task. ACT-R includes a learning mechanism that joins rules that fire close enough to each other (*production compilation)* and strengthening declarative memories through use, so, all the models also learn, with the novice model learning the most and the 100% expert model the least. Table 5 shows the number of initial rules, learned rules through the production compilation process in ACT-R, and declarative memory retrievals at the 1^{st} and the 100th trial.

As shown in Table 5, there are differences among the models in the number of rules and declarative memory retrievals. And these differences in task completion time (as predicted by the ACT-R trace, not wall clock time) are shown in Figure 5.

The 12 kinds of ACT-R models show different times at the first trial; however, the task completion time decreases with practice in all models. Finally, all the models converge at around 400 sec. between trials 40 and 100.

Table 5. The number of initial rules, learned rules, and declarative memory retrievals at the first trial and 100th trial in each model (each declarative memory at Trial 1 has the same amount of the declarative memory elements, 538, related to the keystroke activity).

	Initial rules	Learned rules	DMs used on Trial 1	DM used on Trial 100
Novice	29	253	1,152	1,073
0% Expertise	617	197	1,152	1,036
10% Expertise	617	199	1,091	987
20% Expertise	617	197	1,030	940
30% Expertise	617	199	968	890
40% Expertise	617	199	908	843
50% Expertise	617	199	845	793
60% Expertise	617	196	784	745
70% Expertise	617	199	723	697
80% Expertise	617	199	661	647
90% Expertise	617	198	600	600
100% Expertise	617	197	538	538

Note: The reason that the number of learned rules has slight fluctuations, e.g., 197, 199, 197, 199, etc., is that there is noise in the production compilation process of ACT-R.

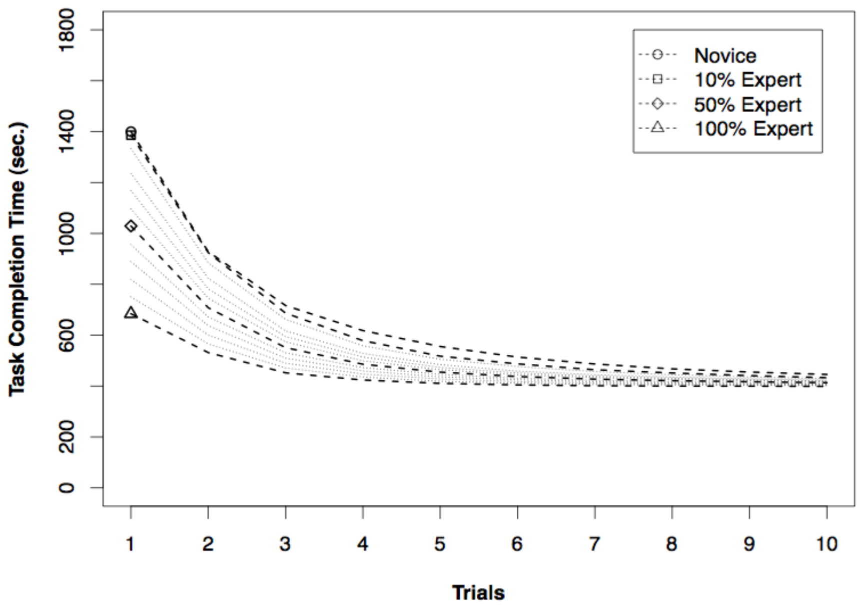


Figure 5. The learning curves for the novice model and the expert models.

5.5 Comparison of the models' predictions with aggregate human data

Figure 6 shows the human data (with SEM as error bars) compared to the prediction times of all the models and also the predicted time for a Keystroke-Level Model (KLM, Card, Moran, & Newell, 1983).

Figure 6 shows that the aggregate human data starts close to the novice and 0% expertise model at the first trial and decreases at the second trial more than both of these models, but within the *SEM* error bars. After the 2nd trial, the human data decreases more gradually. The human data are between the 0% expertise (or novice) model and the 20% expertise model. The best correlation is to the 80% model (r = .999), but all models (but for the 100% expert model) are r > .93 and the smallest RMS error is for the Novice model.

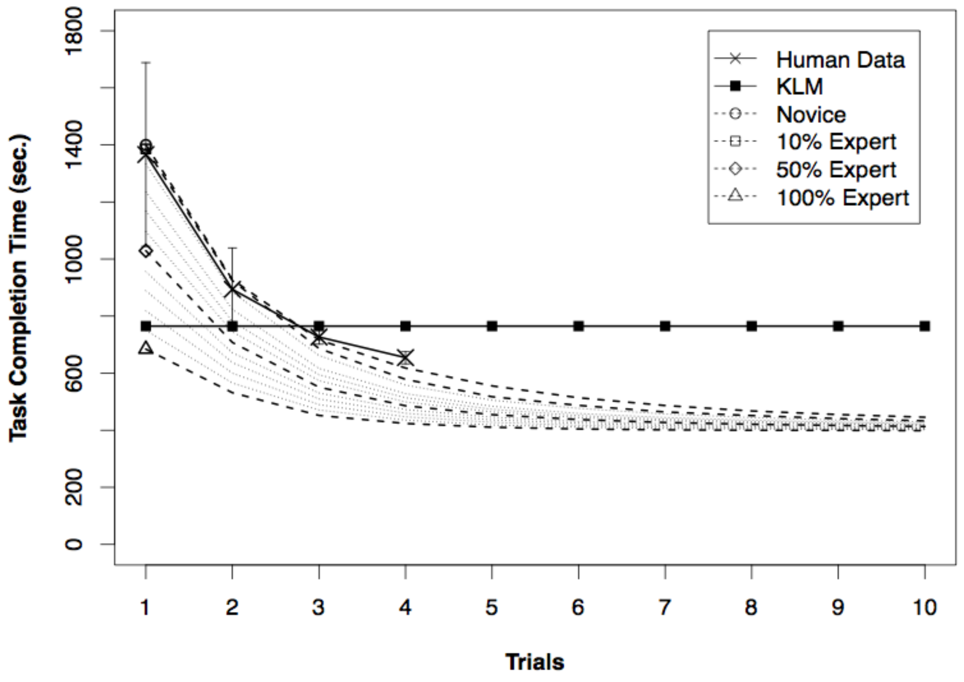


Figure 6. Comparison between the ACT-R models, the KLM model, and the human data on the dismal spreadsheet task. Error bars on the aggregate data are SEM.

The KLM (solid line) predicts experts can complete the task in 765 sec., that is, around the 3rd trial of the human data and for the novice model. For the KLM constants, we use 1.1 sec for the (initial) mouse positioning times, 0.4 sec. for the homing times, 1.35 sec. for the mental preparation times, and 0.6 sec. for the keystrokes. The 0.6 sec. is the average typing speed from 11 participants for this study (Kim, 2008). If we use 0.5 sec. that is the "typing random letters" (see Card et al., 1983, p. 264), the task completion time is 674.6 sec. that is similar to the 4th trial of the human data. However, the KLM does not explain that learning occurs, and it is also the case that the performance of the expert model predicts that users will be faster than a KLM model with the fastest typing speed and with further practice users will move further away from the KLM's predictions. These ACT-R models may be too fast. It has been pointed out that it is more conservative for models to under-predict time (Kieras, 1985). Additional data would be needed to test or for the further development of ACT-R.

Overall, the ACT-R models predict the aggregate human performance from our study. While the human data ends at the 4th trial, we can predict the human learning curve after the 4th trial from the models' performance.

5.6 Comparison of the models' predictions with individual human data

To understand the model better, we compared each subject's performance with the models. Table 6 shows all the model predictions-time correlations. Figure 7 shows the best, worst, and average fits of the models to individual data. Generally, the model fits are high, with the lowest best fit of $r = .845$, lowest worst fit of $r = .832$, and an average best fit of .976. The novice model was the best fit for 17 subjects, the 0% expert model was best for 7 subjects, the 10% expert model for 3 subjects, the 20% expert model for 1 subject, and 100% expert model for 2 subjects.

Table 6. Correlations (r) of the model predictions and subject times. (Best fit noted in the first column and in bold italics.)

	BestFit	Nov	0	10	20	30	40	50	60	70	80	90	100
s7	Nov	***.992***	.987	.989	.987	.988	.988	.990	.989	.989	.988	.988	.988
s9	Nov	***.976***	.968	.973	.971	.970	.971	.973	.972	.972	.972	.972	.971
s11	Nov	***.992***	.987	.989	.988	.988	.988	.989	.989	.989	.988	.988	.988
s15	Nov	***.977***	.970	.973	.971	.972	.972	.974	.973	.973	.972	.972	.971
s16	Exp20	.996	.996	.997	***.998***	.996	.996	.996	.996	.997	.997	.997	.997
s17	Nov	***.968***	.959	.965	.963	.961	.962	.965	.964	.964	.963	.964	.964
s26	Nov	***.960***	.950	.957	.955	.952	.954	.956	.955	.956	.955	.956	.955
s27	Nov	***1.00***	.999	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s29	Nov	***.996***	.993	.996	.995	.994	.995	.995	.995	.995	.995	.995	.995
s30	Nov	***.990***	.986	.987	.985	.987	.987	.988	.988	.987	.986	.986	.986
s31	Exp0	.986	***.991***	.987	.987	.990	.989	.988	.988	.988	.988	.987	.987
s32	Nov	***.976***	.968	.972	.970	.970	.970	.973	.972	.972	.971	.971	.971
s33	Exp0	.996	***.998***	.997	.998	.998	.998	.997	.998	.998	.998	.998	.998
s34	Exp0	.882	***.887***	.880	.876	.887	.883	.882	.883	.881	.879	.878	.877
s40	Exp30	.962	.964	.960	.957	***.964***	.961	.961	.962	.960	.959	.959	.958
s48	Exp0	.960	***.968***	.964	.967	.966	.966	.964	.964	.965	.966	.966	.967
s42	Exp30	.873	.875	.870	.864	***.876***	.871	.871	.872	.870	.868	.867	.866
s43	Exp0	.999	***1.00***	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s44	Nov	***.999***	.999	.999	.998	.999	.999	.999	.999	.999	.998	.998	.998
s45	Nov	***.997***	.995	.997	.997	.995	.996	.997	.996	.997	.997	.997	.997
s49	Nov	***.999***	.997	.999	.999	.998	.998	.998	.998	.998	.998	.999	.999
s54	Nov	***1.00***	.999	.999	.999	.999	.999	.999	.999	.999	.999	.999	.999
s57	Exp40	1.00	1.00	1.00	1.00	1.00	***1.00***	1.00	1.00	1.00	1.00	1.00	1.00
s60	Exp0	.998	***.999***	.998	.998	.999	.999	.998	.998	.998	.998	.998	.998
s63	Nov	***.975***	.968	.972	.969	.970	.970	.972	.971	.971	.970	.970	.970
s64	Nov	***.999***	.998	.998	.997	.998	.998	.998	.998	.998	.998	.998	.998
s69	Exp0	.978	***.984***	.981	.983	.982	.982	.981	.981	.982	.982	.982	.982
s66	Exp20	.836	.832	.839	***.845***	.832	.837	.837	.837	.839	.841	.842	.843
s72	Exp30	1.00	1.00	1.00	.999	***1.00***	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s75	Nov	***.981***	.975	.980	.980	.976	.978	.979	.979	.979	.979	.980	.980

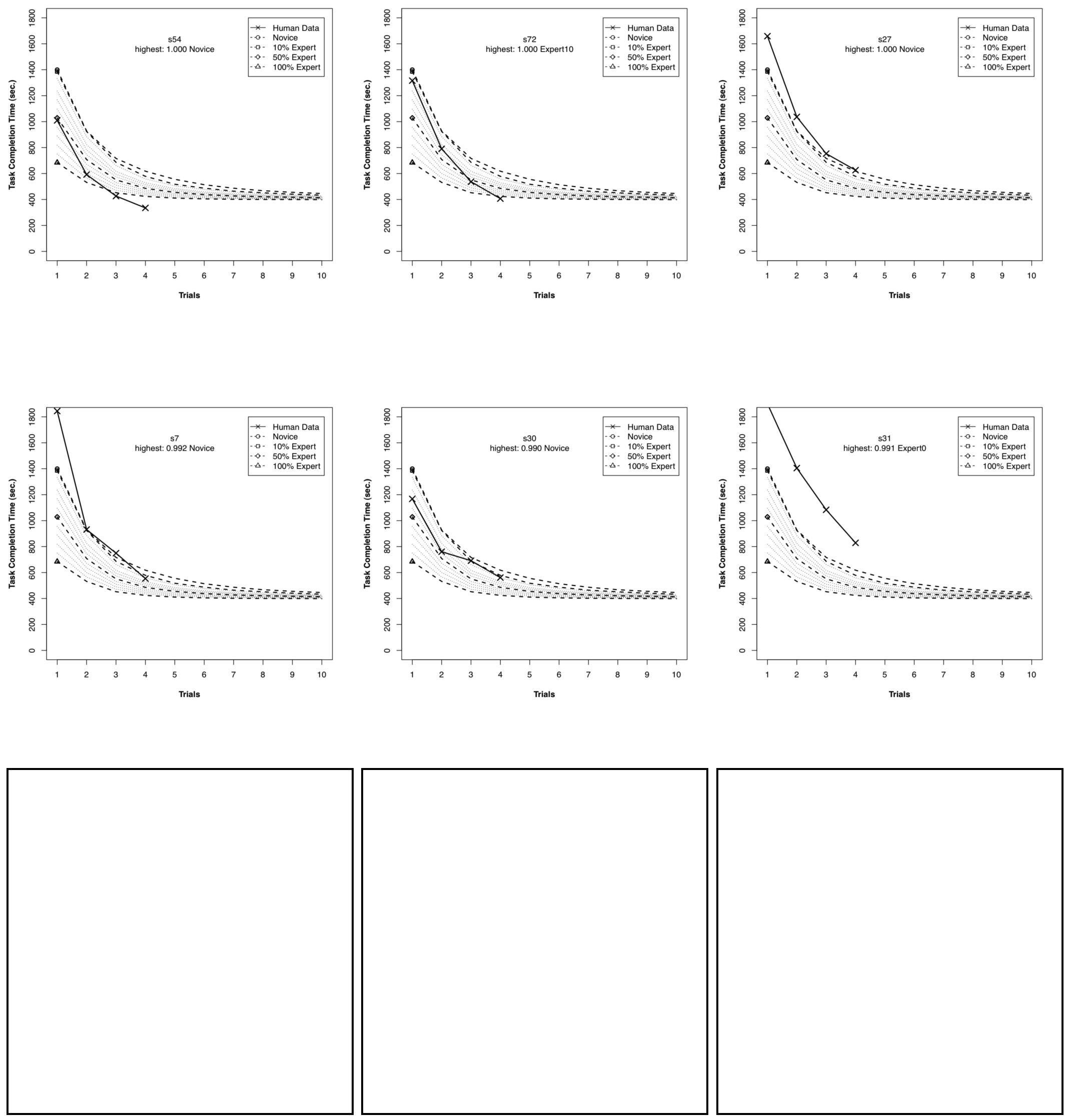


Figure 7. Best (top row), median (middle row), and worst (bottom row) model fits to individual data.

There are some individual differences, but overall, the novice model fits the 30 subjects performance fairly accurately, with a very high r and a relatively small RMS in nearly all cases. The models appear to capture both the average performance and in this case, the individual performances.

6. Discussion and Conclusions

In this article, we briefly reviewed the history of user modeling beginning with GOMS-type engineering models. Although the GOMS-type models have been widely used in HCI for designing user interfaces and predicting human behavior, GOMS-type models have known limitations predicting the performance of novice users and their learning processes. Cognitive architectures such as ACT-R and Soar can predict human behavior more precisely than GOMS models do. However, creating cognitive models of complex HCI tasks is often impeded by their usability. Herbal may present a solution to this concern.

We have developed the Herbal/ACT-R compiler that enables users to decompose a task into hierarchical or sequential subtasks and have this knowledge become proceduralised to generate 12 kinds of ACT-R models: novice, intermediate experts (0% ~ 90%), and normative expert models that perform a complex real world task (i.e., the Dismal spreadsheet task). The models we have developed with Herbal suggest new types of models and new uses for models. One model we noted, Herbal/Soar/Diag includes a large number of strategies (Friedrich & Ritter, 2009). Models of language use will have far more declarative memories about words, but are not likely to have more DMs about task knowledge (e.g., Douglass & Myers, 2010).

Most of the spreadsheet models have 617 rules and the total across levels of expertise is 6,787 rules, and the average number of rules across the 12 models over 100 learning trials is 771. This is also larger than the model reported created with the G2A compiler (St. Amant, Freed, & Ritter, 2005), which had about 100 rules and did not use

ACT-R's learning mechanisms. The model presented here (Herbal/ACT-R/Dismal) is thus perhaps the largest ACT-R model (as measured by rule count) created so far based on the models on the ACT-R website (http://act.psy.cmu.edu/models).

Our models are large, partly because they perform a non-repetitive task. Many previous models designed to perform a task taking minutes execute a repetitive task (e.g., handle 100 airplanes each in the same way). Doing a long non-repetitive task, on the other hand, requires creating a large, complex procedural knowledge set where each piece of knowledge is used less often.

They are also large because they can learn for a long time (up to 100 trials). Across 100 trials, we found that the novice model's task time decreases the most and the expert model's task time decreases the least, which is to be expected. Models at all levels of expertise use less declarative memory over time, which may help them avoid a scaling problem of DM, which can be encountered in ACT-R and Soar (Kennedy & Trafton, 2007).

We also compared those models with human data, where we found that the human data are very similar to the 0% expertise model and the novice model at the first trial and sharply decreases at the second trial (almost to the 10% expertise model). In the third and fourth trials, the curve gradually decreases with practice and ends up between the 0% and novice models again. This result might reveal that humans learn more through the first trial than our ACT-R models, or the variation of the human performance is difficult to predict using the rule learning and declarative strengthening mechanisms of the ACT-R cognitive architecture.

There are some limitations to this work. An improved fit might be achieved by a more precise estimate of the intercept of the model, and perhaps the model not learning as fast. Errors could be examined, as could changes in unit task performance, the time course of the match of model and human performance within a trial, and task retention. Note that the model is simplified in that training and associated proceduralization occurs in four consecutive trials that are not separated by a 23-hour break, as was the case for our human participants. But, despite these limitations these results give rise to several summary comments of interest.

6.1 Going beyond the KLM to model individuals from novices to experts

Comparing the human data with KLM, we found that KLM predicts the task completion time of experts to be around 765 sec., which is about the performance at the 3rd trial of the human data. By adjusting the keystroke time, from 0.6 sec/keystroke (average keystroke time for participants in this study) to 0.5 sec/keystroke (typing random letter), we can get 674.6 sec. for the task completion time of experts, and it is similar to the 4th trial of human data, however, the KLM does not predict the task completion time after the 4th trial of human data nor predict the learning process of our subjects up to that point.

Figure 6 also shows that any model with a fixed task prediction time will have problems predicting expert performance because even expert performance appears to change with learning. From trial 5 to 100 the model's response times decrease. We believe that most users' response times will also decrease, based on learning theory and previous learning data (e.g., Newell & Rosenbloom, 1981; Ritter & Schooler, 2001). The model predicts that the time at trial 100 will vary from 410 to 400 seconds—In any case, the users vary with level of practice and will get faster than the KLM will predict, and will have a distribution of times based on previous learning. Existing tools like the KLM and GOMS do not explain learning or the distribution we see based on previous practice.

6.2 Representing Expertise in Herbal and the Cognitive Complexity Model

The representation of expertise in Herbal differs from the one in the Cognitive Complexity Model (CCM) (Bovair, Kieras, & Polson, 1990). We represented expertise as a function of the model's number of declarative memory chunks and retrievals. Consequently (at least as presently created), the number of retrieved declarative memory chunks gradually decrease as expertise increases, but the numbers of production rules (617 at the first trial) does not differ between the intermediate models and the normative expert model. In the CCM, however, the differences between the novice and expert are represented by the number of production rules to perform a task (Bovair, Kieras, & Polson, 1990). Novice models have more production rules than expert models to complete the same task, and the number of rule firings makes a difference in task completion time between the two. To reduce the number of rules in expert models, CCM assumes practice makes the novice rules set more compact. To compact the rules, CCM excludes rules that are related with checking prompts from the system, and CCM also uses a similar mechanism to production compilation, where rules are composed into a single rule when those rules are always executed in a fixed order.

More concretely, CCMs require modelers to determine which rules can be composed into a single rule by investigating the contents of each rule, then combine those rules to reduce the number of production for representing experts. However, the models generated using Herbal are basically ACT-R models, so they represent that mechanism (production compilation) easily when simulating models in ACT-R because ACT-R's learning mechanisms does the compilation automatically. Herbal also has great efficiency in terms of representing expertise. As we noted in the previous section, we made one very simple procedural task in a sequential/hierarchical manner, then Herbal generates 12 kinds of expertise models that can be simulated in various trials to predict the intermediate level of expertise. However, CCMs require making different models (novice and expert) by reducing the production rules, and cannot predict the intermediate level of expertise.

The basic approaches of CCM for representing novices and experts by investigating which tasks could be and should be learned are not easy to learn for novice modelers (Bovair, Kieras, & Polson, 1990); yet, they are very cognitively plausible. The current version of Herbal does not consider these aspects in generating expertise models. It remains as future work, and it could be used for modeling a wide range of users that vary by expertise.

6.3 Herbal for Rapid Developing of Complex User Models

We have presented a high-level cognitive modeling language that allows for the rapid development of complex user models. As we noted in the introduction, one reason why cognitive architecture user models have not been more widely adopted is perhaps because of the relative difficulty associated with developing them. Cognitive architectures such as ACT-R and Soar use a low-level knowledge representation language that makes developing user models appear intractable to non-experts. Herbal, in contrast, offers a more lucid means of visualizing sequential and hierarchical tasks and creating corresponding models. In addition, Herbal is designed to provide models that explain themselves by providing answers to questions that users frequently ask (Haynes et al., 2009).

6.4 Designing GOMS-like Learning Models with Herbal

GOMS provides the estimated task completion time for the specific system when performed by an expert user, however, GOMS has the limitation that it does not predict the task completion time of novice users and the learning process. We created learning GOMS-like models through the declarative memory pane of Herbal. The ACT-R models created here predict the task completion time of novice, expert, and intermediate users, and they also can provide the learning process of each level of user in each trial through GOMS-like hierarchical task analysis, but one where the model learns to reduce the mental operations and their times.

The results here suggest that for this task users are experts in some way after the 4th practice trial based on the comparison to the predicted time from a simple KLM/GOMS model. The results further illustrate the problem with modeling behavior. With practice, users get faster. Here the KLM predictions are most accurate for trial 4, and then users get faster. (So, we can suggest having users practice a novel task with a known apparatus, i.e., a personal computer, three times for comparison with the KLM model.) But for any value of the KLM, there would be either a mismatch with less expert or more expert users—experts can have a range of expertise, at least in this task, depending on their level of expertise and then with practice.

6.5 Further Understanding of Learning Stages

This model also predicts that the learning behind the learning curve is not smooth on a small scale. This model does not learn the whole task in a completely smooth and uniform way—when it learns, it learns in small steps—the learning happens for particular subtasks within the hierarchy. Each sub task, thus, is not in the same partially learned state, but might vary widely in level of expertise with some subtasks still being in declarative retrieval and some tasks being fully automatized. The learning curve for this model that results when averaging across participants is thus an average of these discrete levels of skill.

A microgenetic analysis of the data might help show how the different sub-skills are in different learning stages, and one might even find that the different skills have different rates of learning or that different sub skills are learned on average earlier or later in the process. This task has several different subtasks, so it would make a reasonable place to start such an analysis.

Understanding when the sub-skills are learned could be useful for building instructional material broadly defined, including the interface itself to support more even learning or to shift critical tasks to more easily or early learned skills. And this knowledge could help build tutors as well, based on knowledge about how subtasks were learned.

6.6 Limitation of the Herbal/ACT-R compiler and future work

We acknowledge that the Herbal/ACT-R compiler is far from mature. It is not yet as easy to use as most end-user environments. It requires familiarity with Eclipse or

XML. The Herbal/ACT-R compiler does not use all of the features of the target architecture, for example, it does not directly include the other learning mechanisms in ACT-R of blending and reinforcement learning.

There remain differences between the support for the different architectures. For example, the declarative model only compiles into ACT-R models. We believe that a Soar compiler could be designed to use declarative memory in Soar, however, differences of learning mechanisms between Soar and ACT-R may pose challenges. Finally, the Herbal/ACT-R compiler uses trees for representing the task, not a graph. This means that when different subtasks are used repeatedly, the model under-predicts the learning that will occur. Herbal still needs more testing with more types of tasks and users that might bring more previous knowledge, but despite these limitations it may be useful beyond this task illustrated here.

There are also limitations that the current Herbal/ACT-R compiler does not address, such as transferring skills across unit task nodes. Users might transfer their knowledge across the subsubtasks that are repeated. However, the current version of the Herbal/ACT-R compiler does not provide any way to represent this transfer. The time gap between the training sessions was also not examined either. These remain as interesting, near future work.

Taatgen proposed Actransfer (2013), which is an extension of the ACT-R cognitive architecture, to explain the transfer of cognitive skills. The study shows that the production rules in ACT-R models can be broken down into primitive information processing elements, which are context-independent units, and these primitive elements can be not only learned in a particular task, but transferred into or used in the other tasks.

The Herbal/ACT-R compiler also uses hierarchical task analysis by decomposing a task into subtasks and unit tasks. This suggests that the Herbal/ACT-R compiler can be further utilized in studying the transfer of cognitive skills.

Taatgen et al. (2008) also studied learning from instruction (Taatgen, Huss, Dickison, & Anderson, 2008) in Flight Management Systems, and argued that list-style instructions are bad for learning and suggested an alternative operator-style instructions that provides not only the steps that participants follow, but also the purpose of the steps are better for learning. Our Dismal task uses list-style instructions only, so it would be interesting we have a different experiment with operator-style instructions and compare performance between two groups. Furthermore, if the Herbal/ACT-R compiler provides an additional component for operator-style instructions, and if we compare the different results from both instructions, it would be of interest.

The errors that participants did while they were performing the task might be interesting to analyze, because errors happen frequently in this kind of task. As we decomposed the task into several subtasks, we can figure out subtasks that error happens more frequently, analyze the reason, and suggest some guidelines.

6.7 *Practical implications of this work*

These results underline our increased ability to model individual users—for some tasks, that is. The models here were created fairly quickly. Even without actually running the models, we believe that they can be helpful at the design stage of user interfaces as a shared representation of the user's knowledge of the task (Pew & Mavor, 2007).

This model provides some general implications for many interfaces, including that users will get faster with practice. Thus, designers should provide them opportunities to practice and base their testing on experienced users. We saw that users in this task become about as fast the KLM predictions after trial 3, so the KLM will provide guidance as well, although the KLM's predictions may be a slightly high for users with extensive practice.

Different user models speed up differently. However, they always end up around the same performance—the greatest variance was found in the first trial. This might well lead to a greater variance in their first impressions. As a consequence, we expect that impressions and evaluations of interfaces and systems would best be taken after some practice. So, to test interfaces with usability studies, one probably should not just have the users perform the task once, but at least several times.

Finally, this model reminds us to that to create easier-to-use and also faster interfaces, the users have to know what to do, that they have to practice doing it, and that regularities and reduced steps help in these areas. For experts, it appears that keystrokes are more costly than the mental representations, which are either automated or removed. With models built with the Herbal framework, we can quantify how much such changes help more broadly and easily.

Acknowledgements. This research was supported by ONR (N00014-06-1-0164, N00014-11-1-0275) and N00014-10-1-0401, and DTRA (HDTRA1-09-1-0054). Mark Cohen, Stephen Haynes, three anonymous reviewers, and readers and reviewers at the HCI Consortium and at the International Conference on Cognitive Modeling provided useful comments on earlier versions of the manuscript.

References

Anderson, J. R. (2007). *How can the human mind exist in the physical universe?* New York, NY: Oxford University Press.

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review, 111*(4), 1036-1060.

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text-editing skill: A cognitive complexity analysis. *Human-Computer Interaction, 5*, 1-48.

Brooks, F. P. (1975). *The mythical man-month: Essays on software engineering*. Reading, MA: Addison-Wesley Pub. Co.

Burton, R. (1998). Validating and docking: An overview, summary and challenge. In M. Prietula, K. Carley & L. Gasser (Eds.), *Dynamics of organizations* (pp. 215-228). Menlo Park, CA: AAAI.

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

Cohen, M. A. (2005). Teaching agent programming using custom environments and Jess. *AISB Quarterly, 120*(Spring), 4.

Cohen, M. A. (2008). *A theory-based environment for creating reusable cognitive models.* Unpublished PhD thesis, Penn State.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2007). Using reflective learning to master opponent strategy in a competitive environment. In *Proceedings of the 8th International Conference on Cognitive Modeling*, 157-162. Taylor & Francis/Psychology Press: Oxford, UK.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2010). Applying software engineering to agent development. *AI Magazine, 31*(2), 25-44.

Cohen, M. A., Ritter, F. E., & Haynes, S. R. (2012). Discovering and analyzing usability dimensions of concern. *ACM Transactions on CHI, 19*(2), Article 9. 18 pages.

Corbett, A. T., & Koedinger, K. R. (1997). Intelligent tutoring systems. In M. Helander, T. K. Landauer & P. Prabhu (Eds.), *Handbook of human-computer interaction*. Amsterdam: Elsevier Science B.V.

Dancy, C. L., & Abuomar, A. M. (2012). *Building a computational adversarial commander model for a warfare simulation*. University Park, PA: Applied Research Lab [unpublished technical report].

Douglass, S. A., & Myers, C. W. (2010). Concurrent knowledge activation calculation in large declarative memories. In *Proceedings of the 10th International Conference on Cognitive Modeling*. 55-60. Drexel University: Philadelphia, PA.

Friedrich, M. B. (2008). *Implementierung von schematischen Denkstrategien in einer höheren Programmiersprache: Erweitern und Testen der vorhandenen Resultate durch Erfassen von zusätzlichen Daten und das Erstellen von weiteren Strategien (Implementing diagrammatic reasoning strategies in a high level language: Extending and testing the existing model results by gathering additional data and creating additional strategies).* Faculty of Information Systems and Applied Computer Science, University of Bamberg, Germany.

Friedrich, M. B., Cohen, M. A., & Ritter, F. E. (2007). *A gentle introduction to XML within Herbal*. University Park, PA: ACS Lab, The Pennsylvania State University.

Friedrich, M. B., & Ritter, F. E. (2009). Reimplementing a diagrammatic reasoning model in Herbal. In *Proceedings of ICCM - 2009- Ninth International Conference on Cognitive Modeling*, 438-439. Manchester, England.

Gray, W. D. (2002). Simulated task environments: The role of high-fidelity simulations, scaled worlds, synthetic environments, and microworlds in basic and applied cognitive research. *Cognitive Science Quarterly, 2*(2), 205-227.

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

Haynes, S. R., Cohen, M. A., & Ritter, F. E. (2009). Designs for explaining intelligent agents. *International Journal of Human-Computer Studies, 67*(1), 99-110.

Haynes, S. R., Kannampallil, T. G., Cohen, M. A., Soares, A., & Ritter, F. E. (2008). Rampart: A service and agent-based architecture for anti-terrorism planning and resource allocation. In *Proceedings of the First European Conference on Intelligence and Security Informatics, EuroISI 2008 (Esbjerg, Denmark, 2008)*, 260-270. Springer: Berlin.

John, B. E., & Kieras, D. E. (1996a). The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction, 3*(4), 320-351.

John, B. E., & Kieras, D. E. (1996b). Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction, 3*(4), 287-319.

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. In *Proceedings of CHI 2004 (Vienna, Austria, April 2004)*, 455-462. ACM: New York, NY.

Jones, R. M., Crossman, J. A. L., Lebiere, C., & Best, B. J. (2006). An abstract language for cognitive modeling. In *Proceedings of the 7th International Conference on Cognitive Modeling*, 160-165. Erlbaum: Mahwah, NJ.

Kennedy, W. G., & Trafton, J. G. (2007). Long-term symbolic learning. *Cognitive Systems Research, 8 (3), 237-247*(3), 237-247.

Kieras, D. E. (1985). The why, when, and how of cognitive simulation. *Behavior Research Methods, Instrumentation, and Computers, 17*, 279-285.

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.

Kieras, D. E., Wood, S. D., & Meyer, D. E. (1997). Predictive engineering models based on the EPIC architecture for a multimodal high-performance human-computer interaction task. *Transactions on Computer-Human Interaction, 4*(3), 230-275.

Kim, J. W. (2008). *Procedural skills: From learning to forgetting.* Department of Industrial and Manufacturing Engineering, unpublished PhD Thesis, The Pennsylvania State University, University Park, PA.

Kim, J., & Ritter, F. E. (2007). Automatically recording keystrokes in public clusters with RUI: Issues and sample answers. In *Proceedings of the 29th Annual Conference of the Cognitive Science Society*, 1787. Cognitive Science Society: Austin, TX.

Kim, J. W., & Ritter, F. E. (2015). Learning, forgetting, and relearning for keystroke- and mouse-driven tasks: Relearning is important. *Human-Computer Interaction, 30*(1), 1-33.

Kukreja, U., Stevenson, W. E., & Ritter, F. E. (2006). RUI—Recording User Input from interfaces under Windows and Mac OS X. *Behavior Research Methods, 38*(4), 656–659.

Laird, J. E. (2012). *The Soar cognitive architecture.* Cambridge, MA: MIT Press.

Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive architectures: Research issues and challenges. *Cognitive Systems Research, 10*, 141-160.

Morgan, J. H., Cheng, C.-Y., Pike, C., & Ritter, F. E. (2013). A design, tests, and considerations for improving keystroke and mouse loggers. *Interacting with Computers, 25*(3), 242-258.

Morgan, G. P., Cohen, M. A., Haynes, S. R., & Ritter, F. E. (2005). Increasing efficiency of the development of user models. In *Proceedings of the IEEE System Information and Engineering Design Symposium*. IEEE and Department of Systems and Information Engineering, University of Virginia: Charlottesville, VA.

Morrison, J. E. (2003). *A review of computer-based human behavior representations and their relation to military simulations* (IDA Paper P-3845). Alexandria, VA: Institute for Defense Analyses.

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

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 1-51). Hillsdale, NJ: Erlbaum.

Newell, A., Yost, G. R., Laird, J. E., Rosenbloom, P. S., & Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Ed.), *Carnegie Mellon Computer Science: A 25-Year commemorative* (pp. 255-293). Reading, MA: ACM-Press (Addison-Wesley).

Olson, J. R., & Olson, G. M. (1990). The growth of cognitive modeling in human-computer interaction since GOMS. *Human Computer Interaction, 5*(2 &3), 221-265.

Pew, R. W., & Mavor, A. S. (Eds.). (1998). *Modeling human and organizational behavior: Application to military simulations*. Washington, DC: National Academy Press. books.nap.edu/catalog/6173.html.

Pew, R. W., & Mavor, A. S. (Eds.). (2007). *Human-system integration in the system development process: A new look*. Washington, DC: National Academy Press. books.nap.edu/catalog.php?record_id=11893.

Pirolli, P. L. T. (2007). *Information foraging theory: Adaptive interaction with information*. New York, NY: Oxford.

Reitter, D., Keller, F., & Moore, J. D. (2011). A computational cognitive model of syntactic priming. *Cognitive Science, 35*(4), 587–637.

Reitter, D., & Lebiere, C. (2010). Accountable modeling in ACT-UP, a scalable, rapid-prototyping ACT-R implementation. In *Proceedings of the 10th International Conference on Cognitive Modeling*, 199-204. Drexel University: Philadelphia, PA.

Ritter, F. E., Baxter, G. D., Jones, G., & Young, R. M. (2000). Supporting cognitive models as users. *ACM Transactions on Computer-Human Interaction, 7*(2), 141-173.

Ritter, F. E., & Bibby, P. A. (2008). Modeling how, when, and what learning happens in a diagrammatic reasoning task. *Cognitive Science, 32*, 862-892.

Ritter, F. E., Haynes, S. R., Cohen, M. A., Howes, A., John, B., Best, B., et al. (2006). High-level behavior representation languages revisited. In *Proceedings of ICCM - 2006- Seventh International Conference on Cognitive Modeling*, 404-407. Edizioni Goliardiche: Trieste, Italy.

Ritter, F. E., & Schooler, L. J. (2001). The learning curve. In W. Kintch, N. Smelser & P. Baltes (Eds.), *International encyclopedia of the social and behavioral sciences* (Vol. 13, pp. 8602-8605). Amsterdam: Pergamon.

Ritter, F. E., Shadbolt, N. R., Elliman, D., Young, R. M., Gobet, F., & Baxter, G. D. (2003). *Techniques for modeling human performance in synthetic environments: A supplementary review*. Wright-Patterson Air Force Base, OH: Human Systems Information Analysis Center (HSIAC).

Rosenbloom, P. S. (2009). Towards a new cognitive hourglass: Uniform implementation of cognitive architecture via factor graphs. In *Proceedings of ICCM - 2009- Ninth International Conference on Cognitive Modeling*, 114-119. Manchester, England.

Schoelles, M. J., & Gray, W. D. (2001). Argus: A suite of tools for research in complex cognition. *Behavior Research Methods, Instruments, & Computers, 33*(2), 130-140.

St. Amant, R., Freed, A. R., & Ritter, F. E. (2005). Specifying ACT-R models of user interaction with a GOMS language. *Cognitive Systems Research, 6*(1), 71-88.

Taatgen, N. A. (2013). The nature and transfer of cognitive skills. *Psychological Review, 120*(3), 439-471.

Taatgen, N. A., Huss, D., Dickison, D., & Anderson, J. R. (2008). The acquisition of robust and flexible cognitive skills. *Journal of Experimental Psychology: General, 137*(3), 548-565.

Yost, G. R. (1992). *TAQL: A Problem Space Tool for Expert System Development.* Unpublished PhD, School of Computer Science, Carnegie-Mellon University.

Yost, G. R. (1993). Acquiring knowledge in Soar. *IEEE Expert, 8*(3), 26-34.

Zhao, C., Paik, J., Morgan, J. H., & Ritter, F. E. (2010). Validating a high level behavioral representation language (Herbal): A docking study between for ACT-R. In *Biologically Inspired Cognitive Architectures, Proceedings of the First Annual Meeting of the BICA Society*, 181-188. IOS Press.