

## Modeling How and When Learning Happens in a Simple Fault-Finding Task

**Frank E. Ritter (ritter@ist.psu.edu)**  
School of IST, 504 Rider, 120 Burrowes St.,  
The Pennsylvania State University  
State College, PA 16801 USA

**Peter Bibby (peter.bibby@nottingham.ac.uk)**  
School of Psychology,  
The University of Nottingham  
Nottingham, NG7 2RD UK

### Abstract

We have developed a process model that learns in multiple ways using the Soar chunking mechanism while finding faults in a simple control panel device. The model accounts very well for measures such as problem solving strategy, the relative difficulty of faults, average fault-finding time, and, because the model learns as well, the speed up due to learning when examined across subjects, faults, and even series of trials for individuals. However, subjects tended to take longer than predicted to find a fault the second time they completed a task. To examine this effect, we compared the model's sequential predictions—the order and relative speed that it examined interface objects—with a subject's performance. We found that (a) the model's operators and subject's actions were applied in basically the same order; (b) during the initial learning phase there was greater variation in the time taken to apply operators than the model predicted; (c) the subject appeared to spend time checking their work after completing the task (which the model did not). The failure to match times on the second time seeing a fault may be accounted for by the subject spent checking their work whilst they learn to solve the fault-finding problems. The sequential analysis reminds us that though aggregate measures can be well matched by a model, the underlying processes that generate these predictions can differ.

### Introduction

There is a speedup effect in the time taken to solve problems that is nearly universal across tasks and subjects (Rosenbloom & Newell, 1987). A major question has been understanding how people this learning occurs. Cognitive models have used several mechanisms to account for learning, such as knowledge compilation and strengthening in the ACT family of models (Anderson, Matessa, & Lebiere, 1998), connection strengthening in PDP models (Rumelhart, McClelland, & group, 1986), rule creation from impasses using analogical reasoning (VanLehn & Jones, 1993), and chunking mechanisms (Feigenbaum & Simon, 1984; Larkin, 1981; Rosenbloom & Newell, 1987). In order to study how multiple learning mechanisms can influence learning, and show how procedural learning may be composed of multiple mechanisms, we explored how to model behavior in a task familiar to us.

There are a number of problems that arise in testing the different approaches to modeling learning. First, the actual path of what is learned and when by an individual learning a

task, to a certain extent, remains predicted but untested. Secondly, there is the problem of the grain size of the measures of subject's performance that have been compared with computational models. Some modelers have focused at the level of problem solving strategy, but detailed comparisons remain rare. Thirdly, some models have automatic learning but few been compared with the time-course of behavior. Doing so would help understand when learning occurs.

These problems make it difficult to compare models of the learning process and have led to a proliferation of different, though equally plausible, accounts of how the time taken to solve problems reduces as cognitive skills develop. This paper will report the results of comparing a process model against subjects' behavior and will address these problems by:

- (a) comparing subjects' behavior individually with the model as they both complete 20 problem solving tasks. Both of their answer times decrease by nearly an order of magnitude across these tasks.
- (b) comparing subject's behavior with the model using several different kinds of measurements.
- (c) using a computational architecture that has an automatic learning mechanism.

We will first explain the task that both subjects and the model completed, and then the model. A comparison between aggregate and individual measures of the model's and subjects' performance follows, which provides support for the model. Because the quality of this match is so high it provides few cues where to improve the model. We then turn to comparing the model's sequential predictions with an individual subject. We conclude by examining the implications of this work on problem solving, multiple learning mechanisms, post-event learning, and the role of aggregate and individual data.

### The Fault-Finding Task

The task that subjects and the model solved consisted of trouble-shooting a control panel device using a memorized schematic. Previous research with a similar task (Kieras & Bovair, 1984; Kieras, 1988) has shown that instructions need to convey to subjects (a) knowledge of the structure of the system, (b) knowledge about the underlying principles that control the behavior of the system, and (c) knowledge about how to perform tasks using the topological and

principle information. Previous work in this area has resulted in summaries of behavior in a series of related tasks such as the role of representations (Kieras & Bovair, 1984), operation of non-faulted circuits (Bibby & Payne, 1993; 1996), and process models that perform multiple tasks but do not learn (Bibby & Reichgelt, 1993).

Ten subjects, all university undergraduates aged 19 to 21 years, were given: (a) a general introduction to the task that included information about power flow through the system and that component indicators light up when the component is receiving power; (b) a picture of the interface that they studied (see Figure 1); and (c) a schematic of the underlying circuit (see Figure 1) showing how the components of the system were connected together by a series of switches, which they were asked to memorize. Subjects were also told that a fault existed when a component indicator light was not lit when it was receiving power. Taken together this information was sufficient to identify faults in the system.

On the interface display and later in the text and figures of this paper, straightforward abbreviations are used to refer to components in the schematic, such as PS standing for power supply and EB1 for energy booster 1.

Once subjects had learned the schematic they were introduced to a computer based version of the system and asked to complete a series of fault finding tasks. Subjects were told that one component in the circuit was faulty and were asked to indicate which component by clicking on it with the mouse. Their reaction times and choices were recorded and analyzed for latency, correct number of choices, and so on.

### The Diag Model

Diag is implemented in the Soar cognitive architecture. The model's behavior is organized around search in problem spaces, and it learns while doing the task. We take these up in turn.

Soar represents problem solving as search through and in problem spaces using operators. When there is a lack of knowledge about how to proceed, an impasse is declared. Soar models will typically end up with a stack of problem spaces, as problem solving on one impasse may lead to further impasses. The stack will change as the impasses get resolved through problem solving. When knowledge about how to resolve an impasse becomes available from problem solving in an impasse, a chunk (a new production rule) is created. This chunk will contain as its condition the knowledge in the higher context that has been used in problem solving in the impasse to recognize the same situation, and it will contain the changes from the problem solving as its action. This simple chunking mechanism, by varying the type of impasse and type of knowledge, has been used to create over 20 complex mechanisms including search control, EBL, and declarative learning, in addition to the mechanisms presented here. Further information on Soar is available at [ritter.ist.psu.edu/soar-faq/](http://ritter.ist.psu.edu/soar-faq/), including online references, pointers to online tutorials, and an extensive bibliography.

Diag has to interact with the task, and there appear to be at least two control mechanisms for interaction that can be used in Soar that support learning. One way is to notice the

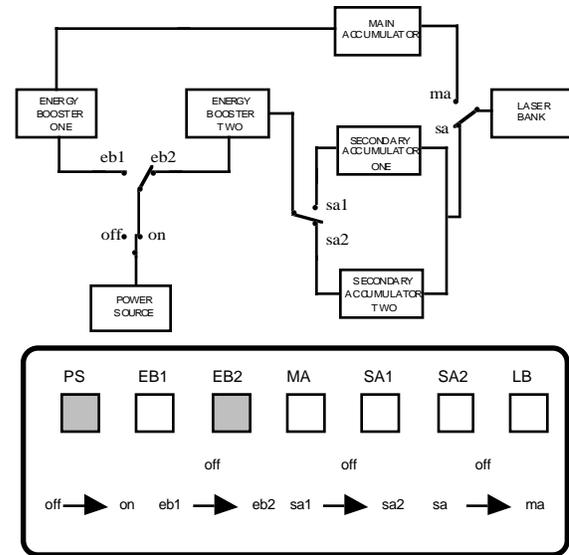


Figure 1. Schematic of the laser-bank device (top), and the laser-bank interface (bottom). Boxes (the lights) below the abbreviations and above the switches (arrows) are grayed out to indicate components are working. This diagram illustrates a faulty component (SA2).

multiple actions as they are proposed and learn to sequence them (Chong & Laird, 1997). Another way is to collapse the goal stack on output to enforce seriality whilst learning what component to attend to, and then to rebuild the stack. This is what Diag does.

Rebuilding the goal stack as an approach to learning interaction makes a testable prediction, that there will be a loss of problem-solving context when initiating an interaction with the environment. This is consistent with the results and review of Irwin and Carlon-Radvansky (1996), who showed that eye movements (which is analogous to the model's interactions) suppress mental rotation and other types of processing, including mental arithmetic and working memory search.

### How the Model Finds Faults

The model begins at the level of minimum competence—the point at which subjects have just memorized the schematic. The model therefore assumes an understanding of power flow through the device, how to recognize a faulty component, and a basic strategy for diagnosing a fault. The combination of schematic and interface knowledge implements a common strategy in this task (strategy #1 in Bibby & Payne, 1996). The present version of the model consists of 186 initial production rules organized in 20 operators in 7 hierarchical problem spaces.

The model does not have immediate access to the entire world state. Rather than view the entire world at once, Diag continually decides where to look (as CHOOSE-COMPONENT), looking there (ATTEND), checking the part it saw (COMPREHEND), and then deciding where to look next (if necessary, again with ATTEND). The organization of components on the (memorized) interface schematic and the use of these operators moving across the

interface representation causes the components to be checked in a basically left to right sequence.

If a subgoal requires an external action to be performed, this external action is passed up as an augmentation to the top state. Knowledge in the top problem space proposes a high priority operator to execute the action. The selection of this high priority operator removes the goal stack, which was in service of some other problem, typically trying to find the fault based on the current information. After the external information is obtained and comprehended, the goal stack is then re-built. The same goal stack may not be rebuilt if the input from the outside world gives rise to a different situation requiring a different response.

Figure 2 shows illustrates the model's performance, showing the operators and their order in solving a fault for the first time. The path illustrates the behavior of the model, particularly the cyclical behavior of choosing a component to check and then checking it. The horizontal lines represent levels in the hierarchical problem spaces.

No attempt to model subject's mistakes was made—incorrect responses were removed from the data used for comparisons with the model. Subjects who scored less than 80% correct were also excluded—this group would be very difficult to model as the assumptions concerning subjects' initial knowledge are harder to justify.

### How and What the Model Learns

Diag learns how to perform actions more directly without internal search and learns which objects to attend to in the interface without internal deliberation. The number of rules learned varies according to the order and number of tasks. On average, the model learns around 200 new rules (chunks) over 20 problems (maximum 222). What Diag learns on each trial depends on the problem and what was learned from previous problems.

What is learned, when it is learned, and how it is learned arises from the architecture, the knowledge and its organization, and the model's problem solving experience. In Diag, these create three types of learning. (a) Operator implementation, where specific knowledge about how to apply an operator is learned through search in a sub problem space. This represents a type of procedural learning.

(b) Operator creation, where an operator is created in one problem space for use in another. The ATTEND operator is elaborated in this way. With practice, the model learns which objects to attend to. These new rules serve as a kind of episodic memory noting the results of problem solving. It is their transfer and application across faults that give the largest improvements.

(c) State augmentation rules, which augment the state with derivable knowledge that act as a type of declarative learning. For example, the model can learn that if EB1 has been checked (and was working) and the current light you are looking at (the main accumulator) is not lit, then the component MA is broken. Each of these types of learning can be seen individually in other architectures and models.

During the course of solving fault-finding tasks newly learned rules can transfer across trials. There is no transfer of newly learned rules within a trial because Diag does not backtrack in its problem solving.

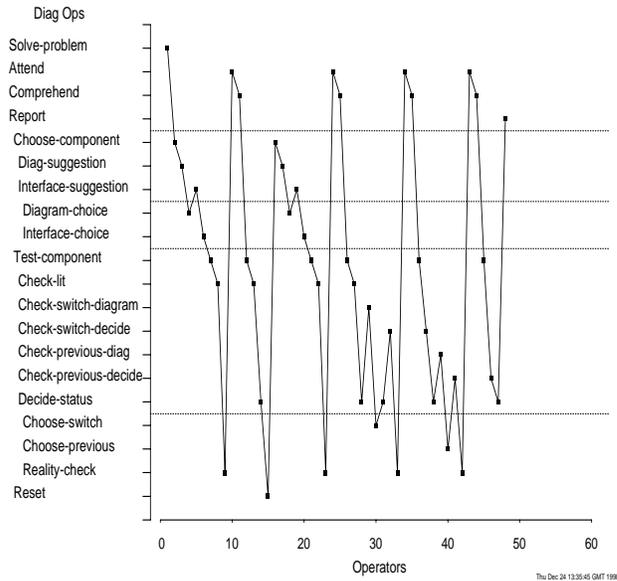


Figure 2. The order of Diag's operator applications the first time it solves the Energy Booster 1 fault.

### When the Model Learns: Comparison with Aggregate Data

Diag accounts for numerous regularities in the data in both the fault-finding task (which we examine here) and the switching task. By matching the transfer between tasks over trials and tasks, we will show that the model matches when subjects learn.

A hierarchical linear regression was used to examine what best predicted problem solving time. The number of components examined predicts 11% of the variability in solution times. Model cycles (with learning off) predicts a further 4%. Diag's decision cycles with learning switched on accounts for an additional 60% of the variability. If the entry of terms is reversed, model cycles of the learning model accounts for all the variance accounted for (75%). The regression and figures to follow suggest that Diag accounts for the subject's behavior because it matches their problem solving *and* their learning.

There is a strong relationship between the time the model and the subjects took to solve the problems ( $R^2 = 0.98$ ,  $p < 0.0001$ ) over task order (Figure 3). As this model suggests, variation in problem solving time is not just due to noise, but much of it can be accounted for by variable amounts of transfer dependent on the current and previous faults. A learning curve is usually a monotonically decreasing curve (Newell & Rosenbloom, 1981) unless task difficulty substantially differs between tasks (Nerb, Ritter, & Krems, 1999); this data is not monotonically decreasing. A post-experiment analysis of the series of faults indicated that our software did not randomly assign faults. Fault 3 was, on average, later in the circuit than the first two faults. To obtain a smooth curve, the average difficulty of faults must remain uniform across the series.

The average variability in problem solving time per subject accounted for by model cycles is 79%. However, the regression was not significant for two of the subjects. When these subjects are removed from the analysis the average variability accounted for increased to 95%. The B coefficient represents the number of seconds per model cycle. S5 has a rate of 10 ms and S7 has a rate of 50 ms. Both these values are significantly lower than the average B coefficient. These subjects do not show the variation in time across tasks like other subjects, and that their performance does not correlate with practice or task type.

Finally, we also examined how the fit varied across trials for the eight subjects for whom Diag's model cycles provided a good fit to their total problem solving times. A typical (but not the best) match between subject and model is shown in Figure 4.

Where Soar problem solving models have been compared with subjects, the model cycle rate is generally longer than 100 ms (Nerb, Ritter, & Krams, 1999; Peck & John, 1992 analyzed in Ritter, 1992), indicating that models are too intelligent, performing the task more efficiently than subjects (taking less cycles because of a more efficient strategy) or not performing as much of the task as the subjects (e.g., not modeling moving the mouse). This is arguably better than being too short, which indicates the model is doing too much (Kieras, Wood, & Meyer, 1997). Models of simple tasks often come quite close to matching data with a 100 ms cycle without adjustments such as reading rate and disambiguating regions of text (Lewis, 1993, p. 201), simple reaction times (Newell, 1990), and covert visual attention (Wiesmeyer, 1991).

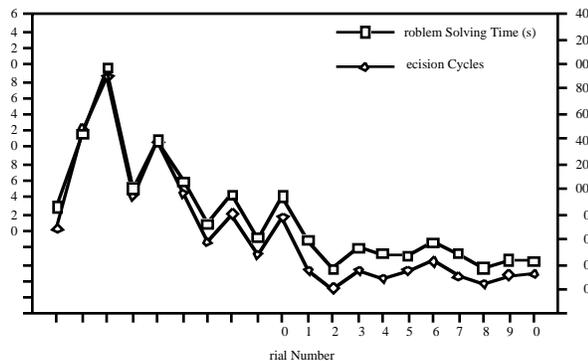


Figure 3. Problem solving time and model decision cycles across trials averaged over all subjects.

### Comparison with Sequential Data

Subjects tended to take longer than predicted to find a fault the second time they completed a task, 32 times out of 47 opportunities. To examine this on a more detailed level, we compared the model's sequential predictions—the order and relative speed that Diag examined interface objects—with an additional subject's verbal and non-verbal protocol while solving five problems. The outcome of this comparison highlighted several places to improve the model and allowed us to identify a new way that the subject learned.

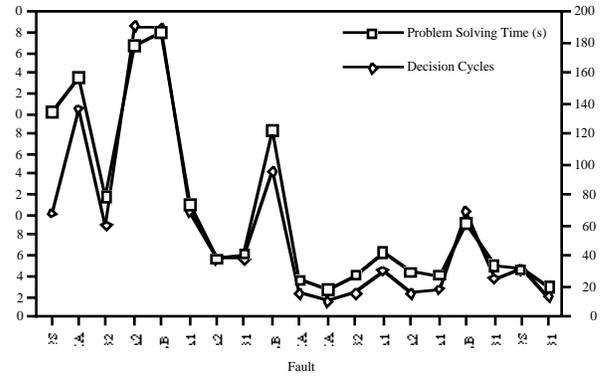


Figure 4. Problem solving time and model decision cycles for individual trials for the fifth best matched subject.

The subject performed the task correctly four times, with performance times comparable with other subjects. While this data may not seem extensive, it represents about 1% of the entire set of verbal protocol data compared with sequential model predictions through 1994 (Ritter & Larkin, 1994).

The model could be aligned with 32 of the 36 codable verbal utterances, and 21 of the 30 codable mouse moves and mouse clicks. The correspondence between subject and model was 91 ms/cycle on average.

Figure 5 shows the time course of how the model's sequential predictions matched the subject's behavior. Overt mouse behaviors are those that can be matched to overt task actions. Implicit mouse behaviors are those movements that can only be compared to internal actions of the model. For example, shifting attention in the model to a component can be aligned to a mouse movement over the same component.

If the model and the subject were solving the task at the same rate, a straight line with unary slope would appear. Deviations from this line indicate that the model and the subject are performing parts of the task at different relative rates, with some typical differences having diagnostic patterns (Ritter & Larkin, 1994). For example, when the slope is greater than 1, the model is performing relatively more work than the subject is.

In Trial 1, top, the subject and the model do not match so well. The relatively horizontal line indicates that interface objects in the subject's early verbal utterances could be matched to objects the model used late in the process, and that the subject spent a considerable time with the solution in mind. The negatively sloped segment around 6 s on the subject's axis indicates that the model and the subject performed two actions in different orders. Trial 2 (2nd from top), shows a slightly better correspondence, and this match improves through Trial 3 (third from top), until in Trial 4 (bottom) the subject and model now perform the task in roughly the same way, at a relatively constant rate.

While the correspondence is good enough to provide additional support for the model, the mismatches are now large enough and systematic enough to suggest that this subject was speeding up in additional ways, through step dropping perhaps.

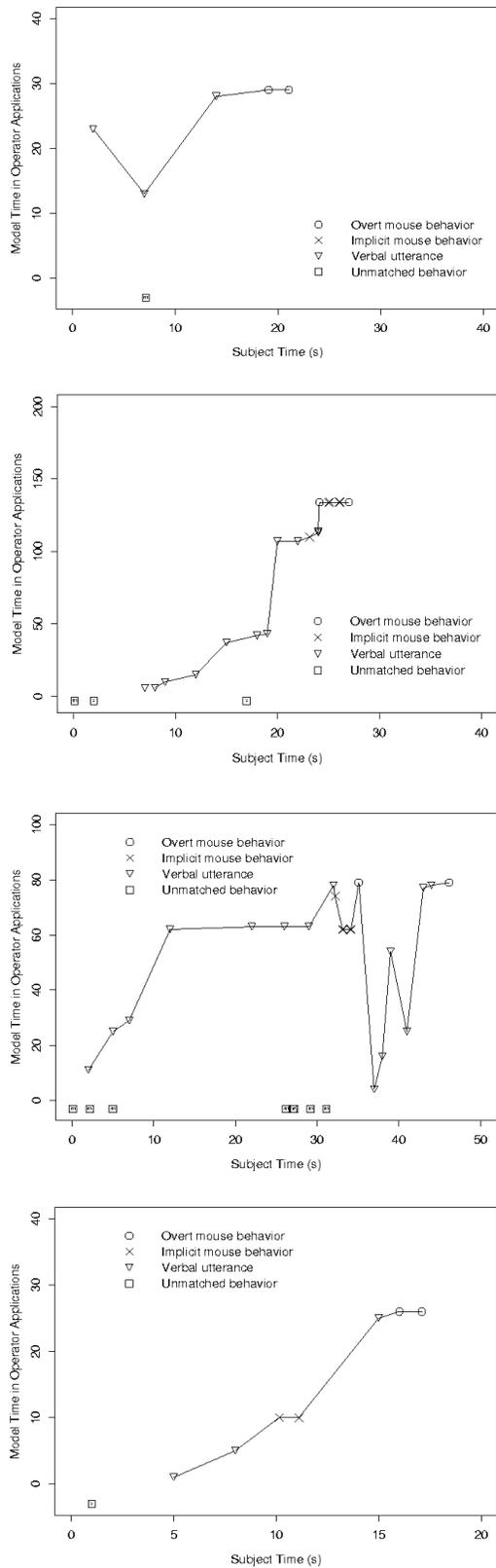


Figure 5. The relative processing rate plots for Trial 1, 2, 3, and 5 (top to bottom), for the protocol subject.

A tentative explanation can also be put forward for the effect where the subject takes a relatively longer time than the model on the second repetition of a task. While the protocol subject did not see a task three times, it appears that the model is learning in a more linear way than subjects, who may be taking time at the end of the second trial to learn in a different way, perhaps reflecting on the problem solving steps.

## Discussion and Conclusions

Diag includes several types of learning. The procedural, declarative and episodic learning all contribute to the speed up on this task and appear to be necessary to predict transfer of learning between tasks. Thus, even though this task takes on average less than twelve seconds to complete, the model predicts that procedural, declarative and episodic learning all play an important part in the speed up in performance of this task. No single kind of learning is responsible for the high level of fit between the model and the subjects' performance. Rather, it is the combination of the different kinds of learning that leads to such a high degree of fit between the model and the data.

Given the high degree of fit between the model and the subjects' behavior failing to implement any of these types of learning would have detrimental effects on the modeling. Diag also interacts with its task, and this too seems essential. These two effects allow Diag to match the strategies that subjects use, the order and components that are examined, the problem solving time, the learning rate, and transfer effects. Overall, while Diag is not perfect, it would be difficult to argue that this model was not a good representation of what subjects do, learn, and transfer in this task.

The close fit of Diag to the learning data also sheds light on the power law of learning. Diag makes a stronger prediction than the power law about learning. Performance does not speed up according to a power law, but according to how much of previous learning transfers to the new task. If the tasks become harder in the course of a series of trials, as shown in Figure 4, then the power law is not obtained. If the amount of transfer truly varies, then the variation in performance, as shown repeatedly in Figure 4, is not noise, but is differential transfer. The power law appears when these details are averaged together, however.

It is quite possible that models in other architectures may be able to duplicate these results. In order to do so, they would have to include the ability to interact with a model of the task. This requires a variety of knowledge, including where to look, how to choose what to believe (internal or external representations), and how to learn through interaction. They must also include learning that leads to the power law in general and support a theory of transfer based on the individual tasks performed.

It looks like Diag leaves out at least two important learning mechanisms (which we can only see by having a learning mechanism in place already), that of decreasing amounts of retracing at the end of a trial, and faster motor output, and perhaps that of faster perceptual recognition (which is likely to exist in this as well as other tasks). These effects are either negligible or are mimicked by and

highly correlated with the existing learning mechanisms. The sequential analysis reminds us that though aggregate measures can be well matched by a model, the underlying processes that generate these predictions can differ.

### Acknowledgments

We thank Ellen Bass, Gordon Baxter, Peter Delaney, Fernand Gobet, Bonnie John, Clayton Lewis, Emma Norling, Mark Steedman, Richard Young and anonymous reviewers for comments and discussions. Sam Marshall provided extensive comments and programmed the model; Shara Lochun assisted with the analysis. Support was provided by the Joint Council Initiative in HCI and Cognitive Science, grant SPG 9018736.

### References

- Anderson, J. R., Matessa, M., & Lebiere, C. (1998). ACT-R: A theory of higher level cognition and its relation to visual attention. *Human-Computer Interaction, 12*, 439-462.
- Bibby, P. A., & Payne, S. J. (1993). Internalisation and the use specificity of device knowledge. *Human-Computer Interaction, 8*, 25-56.
- Bibby, P. A., & Payne, S. J. (1996). Instruction and practice in learning to use a device. *Cognitive Science, 20*(4), 539-578.
- Bibby, P. A., & Reichgelt, H. (1993). Modelling multiple uses of the same representation in Soar. In A. Sloman et al. (Eds.), *Prospects for Artificial Intelligence*. 271-280. Amsterdam: IOS Press.
- Chong, R. S., & Laird, J. E. (1997). Identifying dual-task executive process knowledge using EPIC-Soar. In *Proceedings of the 19th Annual Conference of the Cognitive Science Society*. 107-112. Mahwah, NJ: Lawrence Erlbaum.
- Feigenbaum, E. A., & Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science, 8*, 305-336.
- Irwin, D. E., & Carlson-Radvansky, L. A. (1996). Cognitive suppression during saccadic eye movements. *Psychological Science, 7*(2), 83-87.
- Kieras, D., & Bovair, S. (1984). The role of a mental model in learning how to operator a device. *Cognitive Science, 8*, 255-273.
- Kieras, D. E. (1988). Towards a practical GOMS model methodology for user interface design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction*. North-Holland: Elsevier Science.
- 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.
- Nerb, J., Ritter, F. E., & Krems, J. (1999). Knowledge level learning and the power law: A Soar model of skill acquisition in scheduling. *Kognitionswissenschaft [Journal of the German Cognitive Science Society]*. 20-29.
- 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*. 1-51. Hillsdale, NJ: LEA.
- Ritter, F. E., & Larkin, J. H. (1994). Using process models to summarize sequences of human actions. *Human-Computer Interaction, 9*(3), 345-383.
- Rosenbloom, P. S., & Newell, A. (1987). Learning by chunking, a production system model of practice. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. 221-286. Cambridge, MA: MIT Press.
- Rumelhart, D. E., McClelland, J. L., & group, PDP. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition. Volume 1: Foundations*. Cambridge, MA: The MIT Press.
- VanLehn, K., & Jones, R. M. (1993). Learning by explaining examples to oneself: A computational model. In S. Chipman & A. L. Meyrowitz (Eds.), *Foundations of knowledge acquisition: Cognitive models of complex learning*. 25-82. Boston, MA: Kluwer.