# Action Planning: Producing UNIX Commands

Stephanie M. Doane, Walter Kintsch, Peter Polson

University of Colorado, Boulder

Our goal is to construct a detailed simulation of UNIX user command production based on the construction integration theory of Kintsch, (1988), and building on the action planning model developed by Mannes and Kintsch (1988) and Mannes (1989). The performance we are modeling is that of UNIX users producing legal UNIX commands. Our subjects vary in expertise, and some of the tasks are difficult. But even for the easy tasks, the solutions cannot be precompiled, because familiar elements are put together in a novel fashion. It is presumed that subjects are not recalling fixed scripts from memory. Rather, they are producing action plans for each task.

## THEORETICAL BACKGROUND

Subjects are presented with their task through a textual description of the goal. For example "sort the first ten lines of file x alphabetically". We are simulating user understanding of this instruction and the generation of appropriate plans by extending the Mannes and Kintsch (1988) planning model which is based on the discourse comprehension theory of van Dijk and Kintsch (1983), and Kintsch (1988).

Mannes and Kintsch (1988) constructed a simulation of subjects doing routine computing tasks involving a file and mail system. They built their knowledge base from the results of verbal protocols, where subjects verbalized how they would accomplish given tasks (Kintsch and Mannes, 1987). Their model simulated retrieval of relevant knowledge using instructional text as cues, and it showed how this led to the formulation and execution of action plans.

van Dijk and Kintsch (1983) and Kintsch (1988) make the point that comprehension of text that describes a problem to be solved (e.g., an algebra story problem) involves formulating an effective solution plan for that problem, retrieval of relevant factual knowledge, and utilization of appropriate procedural knowledge (knowledge of algebraic and arithmetic operations). In Mannes and Kintsch (1988) this is modeled by retrieving both relevant general knowledge and specific knowledge about files and editing tasks, and then formulating editing action plans. Thus, understanding an editing task means generating a mental representation of the elements of an editing system (e.g., files) and the relationships between those elements, and then using this information to accomplish the task.

Our goal is to determine what knowledge about UNIX the model requires to produce legal UNIX commands. The fundamental assumption underlying the UNIX user interface is that complex commands can be created by concatenating simple commands with the use of advanced features that redirect command input and output (e.g., pipes). We wish to understand what kinds of knowledge are necessary to use these advanced features (e.g., input/output redirection). We are also interested in the actions themselves; part of producing a UNIX command includes making a plan for action, but knowing the specific action (e.g., knowing the command "ls" lists file names) is critical to plan execution. Thus, we are extending the Mannes and Kintsch work to include actions, in a manner analogous to work of Card, Moran and Newell (1983) and Kieras and Polson (1985).

Unlike Mannes and Kintsch, we are starting from a rational analysis of knowledge necessary to perform tasks within the UNIX system, and performance data, rather than from verbal protocols. As such, we have attempted to develop a performance model based on our assumptions about the knowledge of UNIX required to produce legal command syntax.

## PERFORMANCE DATA

Doane, Pellegrino, and Klatzky (1989, in press) examined the development of expertise within the UNIX operating system by studying UNIX users with varying levels of experience. A portion of their research measured users' performance in tasks requiring them to produce UNIX commands. Included in their studies were both longitudinal and cross-sectional analyses of the emergence of command production expertise.

In this research we are attempting to model a portion of the production performance measured in the Doane et. al., (in press) research. In the production task, subjects were asked to produce the most efficient (i.e., the least number of keystrokes) legal UNIX command that they could to accomplish a specified task. Tasks ranged in difficulty from individual, frequently used UNIX commands to composite commands that effected several actions that had to be sequenced appropriately using pipes or other input/output redirection symbols. The tasks were designed to assess the impact of two types of knowledge; knowledge of individual commands and knowledge of the processes involved in sequencing those commands properly. Tasks involving more elementary commands were designed to include elements that had to be put together in order to generate a successful composite command.

An example composite task would be "display the first ten alphabetically arranged file names of the current directory". A component single would be "display the file names of the current directory". A multiple would be "display the file names of the current directory", "arrange the contents of file x alphabetically on the screen", and "display the first ten lines of file y". Thus, "single" commands caused just one action, multiple commands caused several independent actions and composite commands caused several actions that had to be sequenced appropriately using pipes and/or redirection symbols. These were equal in length to the multiple commands; thus the essential difference between the two was that composites had dependencies among their components whereas multiples did not.

The production data indicate that UNIX users differ markedly in performance, according to their history of use with the operating system. The striking aspect of these data was that only the experts could successfully produce the composites, even though the intermediates and novices could perform the other tasks that were designed to assess the component knowledge required to successfully generate a composite (e.g., singles). This is somewhat surprising, since the knowledge necessary to perform the composites is taught to intermediates and novices. The novices and intermediates had, on the average, 8 months and 2 years of experience with the system, respectively. And all of the subjects had taken a course which taught about pipes and other redirection symbols, and required their use for course homework. Thus, the less expert groups have the knowledge, but can't use it productively in the sense discussed by Wertheimer. (1982/1945).

To summarize the Doane et al. (in press) data, novice and intermediate UNIX users appear to have knowledge of the elements of the system, they can successfully produce the single and multiple commands that make up a composite. They could not, however, put these elements together using pipes and/or other redirection symbols to produce the composite commands. As previously stated, the symbols that enable input/output redirection are fundamental design features of UNIX, and these features are taught in elementary classes. Doane et al (in press) demonstrate that these features can only be used reliably after extensive experience (e.g.,experts had, on the average, 5 years of experience with UNIX).

## RESEARCH GOALS

One of our goals is to understand in detail the kinds of knowledge that are sufficient to make effective use of these advanced features. To simulate expert performance on composites, the model requires more knowledge than is necessary to perform singles and multiples. In addition to knowledge of the basic types of commands and their syntax (the only knowledge required to execute singles and multiples), the model must have knowledge of the ordering of commands in a sequence. For example,to execute the task "display the first ten lines of the alphabetically sorted file x",the model must know that the command that sorts files alphabetically (SORT) should be executed on file x prior to executing the command that will display the first ten lines of the file (HEAD). The model must also know about the specific redirection properties of each command. In the example task above, the model must know that the output of SORT can be redirected to another command, and that the input to HEAD can be redirected from another command.

A possible application of such research is to guide instruction How would you modify instruction to novices to assist their performance? Another application is to guide system design. By modeling performance, we hope to gain some insight into the attributes of UNIX system design that may hinder acquisition of expertise. This may provide some guidance toward designing a system which would provide similar advances facilities without demanding such strenuous knowledge prerequisites.

We are trying to simulate the solution of tasks that cannot be precompiled, because while the elements of some of the tasks may be familiar, and indeed overlearned, they are often put together in novel sequences. That is, we are not dealing with fixed scripts which can be retrieved from memory, but with plans of action that are constructed in the context of a specific task. The construction - integration model is an appropriate model for our goals. It is a comprehension model of problem solving that does not assume a highly structured precompiled knowledge.

## THE MODEL

The construction - integration model proposed by Kintsch (1988) contains four main components. The first two components are representations of knowledge; a propositional textbase derived from the instructional text, and an associative long-term memory. The next two components are processes which activate the represented knowledge; an activation process which allows text propositions to activate information in long-term memory, and a final integration process which selects relevant knowledge, and deactivates irrelevant knowledge. These processes are repeated cyclically. When some action is executed, the state of the world is changed to include the result of that action. The model continues processing in cycles until the desired state of the world is achieved (the task is accomplished), or until the system fails. Described below are the elements that together form a simulation of UNIX user's performance. Portions of the description of the model's calculations are a summary of discussion found in Mannes and Kintsch (1988).

### Long-term Memory

Eighty-eight propositions were constructed from our rational analysis about the knowledge of UNIX required to execute 29 of the UNIX production tasks from the Doane et al. (in press) experiment. These 29 tasks were made up of different combinations of 9 single commands. There were 9 single tasks, 8 multiple tasks, and 12 composite tasks. (These 29 tasks are a small subset of the tasks used by Doane et al. .) These propositions were used to simulate a portion of the user's long-term memory (LTM) network.

As mentioned above, we are interested in understanding the knowledge required to use the advanced features of UNIX. This knowledge is represented in three main forms in the LTM portion of the knowledge base. First, the LTM contains knowledge of the atomic commands that one can perform (e.g., print a file). Second, it contains knowledge of the redirection properties of these commands required to combine them into composite commands. For example, in order to execute the action LS|LPR, a modeled user must know that the output from LS can be redirected to another command, and that the LPR command will take input from another command. Third, the LTM contains knowledge of the syntax required to produce a command sequence (e.g., knowing that LPR prints a file, and that the pipe symbol redirects input and output).

Interconnections for this network were calculated based on argument overlap, and proposition embedding. Additive connection strengths of .7 were used for each case of argument overlap or embedding. A LISP program computes the interconnection values among all the items in long-term memory, creating a nxn connectivity matrix. The connectivity matrix is intended to approximate an association matrix. We recognize that this relatively simple and objective way of estimating values provides us with crude and fallible measures of association. The next portion of the long-term memory consists of 11 plan elements, 9 of which correspond to the single building-block commands, and 2 of which are plan elements that allow creation of new plans. These latter plan elements are called building plan elements, because they allow the modeled user to build a composite command from the single building-block commands.

Three of the single plan elements, and one of the build plan elements are shown, with their text in abbreviated form in Table 1. There are three components in each plan, a plan name, preconditions, and outcomes. The preconditions are propositions which represent states of the world which must exist for the plan to be executed. The outcomes are propositions which become states of the world if the plan is executed. The plan names are also propositions.

Connection between propositions in the long-term memory net and the plans are based on certain types of propositions which Mannes and Kintsch call REQUESTS and OUTCOMES. Requests are imperative verbs that are used in task descriptions, requesting that the user accomplish a task (e.g., sort a file, print a file). The relationship between these verbs and plans is more precise than that given by argument overlap. Each REQUEST proposition must be associated with a particular plan or set of plans. For example, the REQUEST DISPLAY FILE NAMES DIRECTORY is associated directly with the plan (DISPLAY FILE NAMES DIRECTORY), and with other plans that are considered to be DISPLAY plans (e.g., DISPLAY FIRST-TEN LINES FILE) with a connection value of +1, and with all other plans with a connection value of 0.

Each OUTCOME proposition is associated with a +1 value to the plan that produces this outcome, with a value of -1 to plans which produce an incompatible outcome (e.g., the outcome CREATE FILE X and a plan which DELETES FILE X), and with a value of 0 to plans which produce an irrelevant outcome (e.g., the outcome CREATE FILE X and a plan to DELETE FILE Y). Only final goals are connected in this manner. Plans which need to be executed before the final goal can be executed must be contextually activated through the network activations.

### Textbase

A propositional textbase was derived from the text of the problems posed to subjects (e.g., display the file names of the current directory) following the methods outlined in Kintsch (1985). Each textbase proposition activates two other associated propositions in the long-term memory net. The exact computations are described in Mannes and Kintsch (1988). As before, the request

**Table 1. List of four plans.**

| Plan Name | Preconditions | Outcomes |
|---|---|---|
| (DISP FL DIR) | (@SYS)(EXIST DIR) (KNOW LS) | (DISP FL DIR) |
| (SORT FL) | (@SYS) (EXIST FL) (KNOW SORT) | (SORT FL) |
| (DISP 1ST TEN LINES FL) | (@SYS) (EXIST FL) (KNOW HEAD) | (DISP 1ST TEN FL) |
| (BUILD PIPE) | (KNOW LS 1ST) (DISP FL DIR)<br>(KNOW SORT 2ND) (SORT FL)<br>(KNOW HEAD 3RD) (DISP 1ST TEN FL)<br>(KNOW REDIRECT HEAD OUTPUT)<br>(KNOW REDIRECT SORT INPUT)<br>(KNOW REDIRECT SORT OUTPUT)<br>(KNOW REDIRECT HEAD INPUT) | (USE PIPE PLAN) |

propositions require special treatment. They do not randomly activate associated knowledge, but must be used to retrieve a particular outcome. Specifically, REQUESTS and OUTCOMES are used as a joint retrieval cue to retrieve the outcome of the appropriate request -- which is associated with both retrieval cues, as in Raaijmakers & Shiffrin (1981) and Kintsch and Mannes (1987). In summary, the model now contains the text propositions along with their associates and their outcomes. When the model is trying to accomplish a task, all plans must have the potential of being activated. Thus, to the text propositions we add all 11 plan elements.

A matrix is constructed from this textbase, and the interconnections between items are calculated in the same manner as it was for the long-term memory matrix, with three additional calculations. The first calculation concerns the relationships between outcomes and plans. If all of the outcomes of a plan currently exist in the current world of the model, they inhibit the firing of that plan. For example, if the FILE X is already printed, the plan (SEND FILE X TO THE LINE PRINTER) will be inhibited. The next two calculations concern the relationships between the plans themselves. Expected outcomes of plans are related to each other in the same way as in the long-term memory matrix. That is, plans can activate other plans which produce compatible outcomes, and inhibit plans which produce incompatible outcomes. Finally, plans themselves are interconnected by a causal chaining mechanism. For example, a plan that requires that FILE X exist will activate plans that have the existence of FILE X as an outcome. Thus, Mannes and Kintsch (1988) designed the model so that interconnections among plans represent the user's knowledge about causal relationships within, in this case, the UNIX operating system.

The connectivity matrix of size nxn is obtained for each task description, corresponding to the original m text propositions and to the (n-m) associates, outcomes, and plans that have been added to the text. An initial activation vector with n elements is then constructed, with activation values 1/m for the m original text propositions and 0 for all others. This vector is then repeatedly postmultiplied with the connectivity matrix, with the m values in the vector reset to 1/m and the remainder of the vector renormalized after each multiplication. This multiplication computes the spread of activation among the elements of the vector, with the reset of the m elements trapping the activation value of the in the world propositions such that it prevents weakening of their effects with each iteration. When the change in the activation vector between iterations reaches an arbitrary criterion of .0001, the activation pattern is assumed to reflect the stable state of knowledge formed by the modeled user given the current task.

To summarize, in terms of the van Dijk and Kintsch model (1983), our small textbase activated both irrelevant and relevant knowledge. The model then used an integration process to spread activation to the relevant knowledge, and thus to simulate the modeled user's state of knowledge given the current task.

### Plans and Action

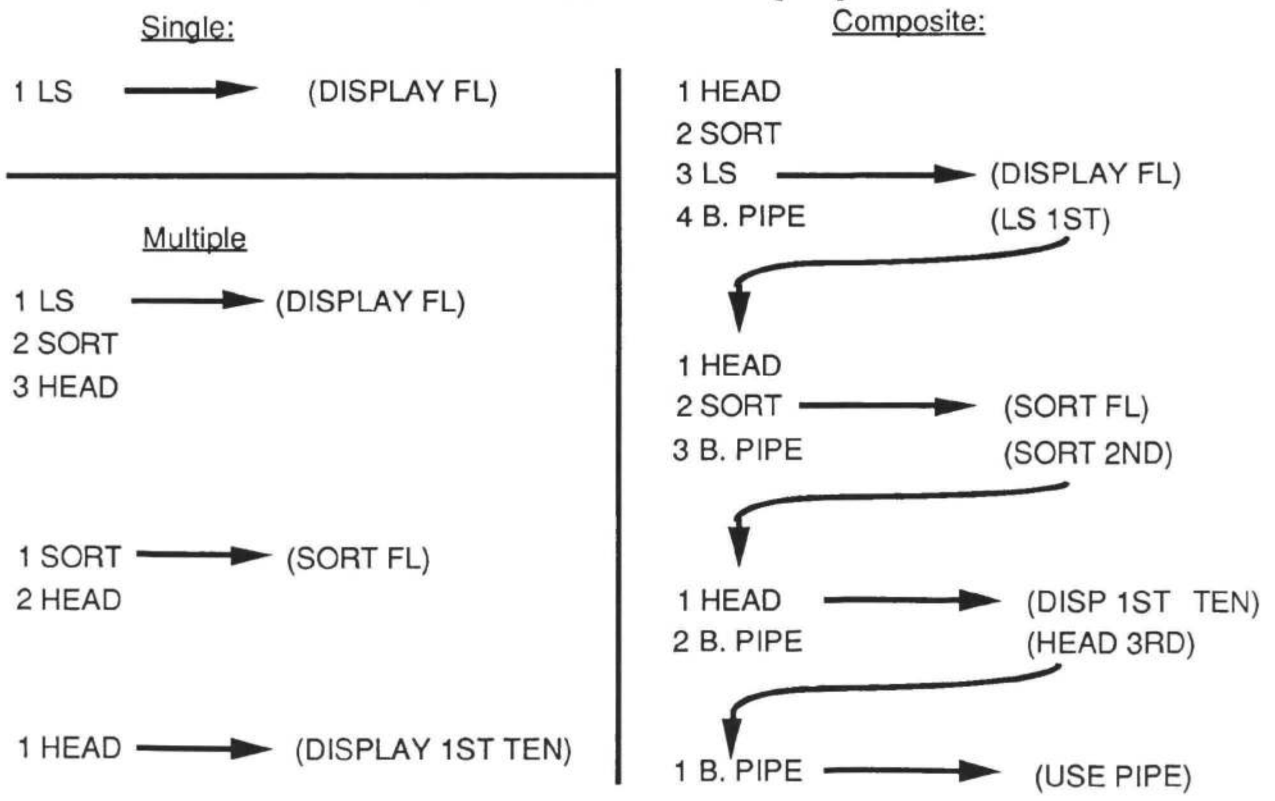
The model of the user's state of knowledge results in plans with different levels of activation. Mannes and Kintsch (1988) propose an executive process which examines plan activation values, and picks for execution the plan with the highest level of activation. If the preconditions of that plan are met (the preconditions exist in the world), then the plan is fired. However, if the preconditions are not met, then the executive process goes to the next most highly activated plan, and so on. If the outcome of the executed plan is the expected outcome, then the task is complete. If not, then the outcome of the executed plan is added to the model of in the world knowledge, and the integration process is repeated, resulting in a new pattern of plan activation. The process described above is reiterated until the plan with the expected outcome fires.

## THREE EXAMPLES

Table 2 depicts the main results of simulating the component tasks shown in Table 1 in the form of singles, multiples and a composite. The table shows only those plans considered by the executive process, along with their relative activation values. The simulation of the single command to display the file names of the current directory (LS problem) is very simple for the model, as it was for the subjects in Doane et. al. The system finds all of the preconditions are met, and the plan is executed immediately. For the multiple problem (LS, SORT, HEAD), the executive first chooses the LS task, its preconditions are met, and its outcome (the proposition (DISPLAY FILE-NAMES) is added to the in the world knowledge. After a new integration phase, the most activated plan is SORT. The executive chooses to fire the SORT plan, and its outcome (SORT FILE) is added to the knowledge base. Finally, the same process occurs for the plan HEAD, and the task is complete.

The composite problem (to produce LS|SORT|HEAD) is far more complex, since the task requires that the model go thru both a planning phase and a building phase. In the single and multiple tasks, there are no interrelationships between sequences of actions, and the model immediately generates appropriate user actions upon determination of the correct plan. In the case of the composites, we argue that the appropriate plans must be determined, and then sequenced correctly using the interrelationships among the preconditions for the various plans. Following each integration phase, this planning process adds representations of the individual commands and the order in which they should be executed to the in the world knowledge. In our example, the

**Table 2. Traces of solutions to example production tasks.**



request which is acted on first is LS; the system would like to execute the HEAD plan, but its preconditions have not been met. The preconditions for HEAD in this task require the existence of an alphabetically sorted listing of the file names in the current directory. That is, the (EXIST FL) precondition has been bound to a specific file; the required file contains the alphabetically sorted file names from the current directory. The next most highly activated plan is SORT, but it has the precondition that a display of the file names exist. The (EXIST FL) precondition is now bound to a file containing the file names of the current directory. The only plan with satisfied preconditions is LS. It is fired, and the outcome (DISPLAY FILE-NAMES) is added to the current knowledge base, along with a representation of its order in the task sequence (i.e., that it is first). After the next iteration, the simulation would again like to fire the HEAD plan, but can't. The preconditions for the SORT plan are now met, and it fires, and the outcome (DISPLAY SORTED FILE-NAMES) is added to the knowledge base, as is the information that it is second in the sequence, following LS. On the final planning iteration, the HEAD plan is again the desired plan, its preconditions are now met, the plan is fired, relevant in the world knowledge is added, and the planning phase is complete.

Following the planning phase, these new representations are used by the build pipe plan to create the actual composite command. The model will execute the build pipe plan when its preconditions are met, but the preconditions for the build pipe plan are extensive. In order for the plan to fire, the modeled user must have, for each command, knowledge of the command syntax, knowledge of the command order in the sequence, and knowledge of the command redirection properties. Since we are modeling the expert, the system has all of the prerequisite knowledge, and the build pipe plan fires. The outcome of this plan is the creation of a "use pipe" plan, which is bound to the syntactically correct sequence of commands. When the use pipe plan fires, the task is complete.

While on the surface, the construction of a composite command seems simple, the psychological processes involved appear to be very complex. The creation of a composite command appears to involve the use of complex knowledge in planning and building activities. Our planning phase suggests that it should take longer to begin composing a composite command than to begin composing a single or multiple, especially for the experts. In fact, this is exactly the pattern of response time reported in Doane et. al., (in press). Experts took significantly longer to make an initial keystroke when composing composites than when they were composing less complex commands. This pattern was not found with novices, which suggests that they did not complete a satisfactory planning phase. In addition to the knowledge prerequisites, the planning phase for construction of composites seems to make a large demand on working memory. Previous researchers have suggested (e.g., Anderson & Jeffries, 1985) that novice performance may suffer due to loss of information from working memory. Perhaps one of the reasons that novices don't succeed in constructing composite commands is that planning these tasks create a large working memory load.

## WORK IN PROGRESS AND FURTHER DEVELOPMENTS

The ability of the construction - integration model to simulate the production processes for the tasks considered here is encouraging, though our results are not yet definitive. The simulations described above model the ideal expert performing UNIX command production tasks. One of our major goals is to systematically break down the knowledge base in a manner that explains UNIX performance for users at different levels of expertise. That is, we want to determine what aspects of the knowledge base must be deleted to simulate intermediate and novice performance. This will allow us to simulate the nature of UNIX knowledge acquisition, perhaps even on the level of individual differences. The longitudinal data from the studies provide us with an initial starting point for this task. We will need to develop a methodology for comparing the details of model predictions to the empirical data. In our current comparisons, we have noted good correspondence.

Another goal is to analyze the results of these simulations, such that we can make some recommendations about teaching UNIX skills, and about system design. All that we have noted so far is that use of the advanced features of UNIX requires a good amount of knowledge above and beyond knowledge of the command elements themselves

## CONCLUSIONS

Though this work is currently incomplete, we are encouraged by our progress. As in Mannes and Kintsch (1988), what has been accomplished is not in itself surprising: we are using a means-end, backward problem solving mechanism similar to mechanisms included in the General Problem Solver. Unlike previous problem solving models, we are using this mechanism in the context of a general theory of discourse comprehension. We are extending the work of Kintsch and Mannes (1988), which adapts this comprehension model to the problem solving domain by representing the domain specific knowledge as plans.

## REFERENCES

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

Doane, S. M., Pellegrino, J. W. and Klatzky, R. L. (1989). Mental Models of UNIX as Revealed by Sorting and Graphing Tasks. *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, Kailua, Kona, Hawaii: IEEE Computer Society.

Doane, S. M., Pellegrino, J. W., & Klatzky, R. L. (in press) _Expertise in a computer operating system: Conceptualization and performance. *Human-Computer Interaction*.

Anderson, J. A., & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human-Computer Interaction, 1*, 133-161.

Kieras, D. E., and Polson, P. G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies, 22*, 365-394.

Kintsch, W. (1985). Text processing: A psychological model. In T. A. van Dijk (Ed.), *Handbook of Discourse Analysis*, Vol. 2 (pp. 231-244). London: Academic Press.

Kintsch, W. (1988). The use of knowledge in discourse processing: A construction-integration model. *Psychological Review, 95*, 163-182.

Kintsch, W. & Mannes, S. M. (1987). Generating scripts from memory. In E.vanderMeer & J. Hoffman (Eds.), *Knowledge aided information processing* (pp. 61-80). Amsterdam: North-Holland.

Mannes, S. M. (1989). *Problem-solving as text comprehension: A unitary approach*. Unpublished doctoral dissertation, University of Colorado, Boulder.

Mannes, S. M. & Kintsch, W. (1988). Action planning: Routine computing tasks. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 97-103). Montreal, Quebec, Canada: Erlbaum.

Raaijmakers, J. G. & Shiffrin, R. M. (1981). Search of associative memory. *Psychological Review, 88*, 93-134.

van Dijk, T. A. & Kintsch, W. (1983). *Strategies of discourse comprehension*. New York: Academic Press.

Wertheimer, M. (1982/1945). *Productive thinking*. Chicago, IL: University of Chicago Press.