Near-term memory in programming: a simulation-based analysis

ERIK M. ALTMANN

Department of Psychology, Michigan State University, East Lansing, MI 48824, USA.
email: ema@msu.edu

Near-term memory (NTM) is proposed as a construct for analysing the memory that experts build up and use as they solve a problem in their domain of expertise. Large amounts of information are processed in such situations, and any particular detail could become important later, so performance is facilitated by maintaining long-term memory access to as much detail as possible. Precise analysis of such memory is difficult to achieve with experimentation or observation alone, so computational simulation is used as the analytical method. A computational process model grounded in cognitive theory (Soar) is constructed to fit extensive fine-grained behavioral data from an expert programmer. The model's structures and processes are then inspected for insights into NTM. Structurally, the model's NTM consists of fine-grain perceptual, semantic, and episodic items whose availability is tied to cues from the encoding context. Quantitatively, much more detail enters NTM than is ever retrieved, but when retrieval does occur it can change the course of behavior. To illustrate applications of the construct, the model is used to examine how a cluttered interface might impose cognitive costs by increasing retrieval demands on memory.

© 2001 Academic Press

1. Introduction

Expertise in programming has been quantified in terms of tens or hundreds of thousands of rules making up a programmer's domain knowledge (Brooks, 1977), consistent with estimates of tens of thousands of chunks comprising expertise in other domains like chess (Chase and Simon, 1973). However, such analyses of expertise emphasize long-term, semantic knowledge, built up over years, and not only through simple practice but through deliberate reflection and inference (Ericsson & Lehmann, 1996).

A largely unexamined aspect of expertise in programming is the information accumulated in episodic memory moment-to-moment as performance proceeds. Various studies suggest that people are faced with the need to store in memory large amounts of detail as they perform a complex task (Jeffries, Turner, Polson & Atwood, 1981; Ericsson & Kintsch, 1995; Altmann & John, 1999). Moreover, the lag between encoding and needing any particular item could range from seconds to minutes or perhaps hours, placing a premium on retention. Finally, the likelihood of needing a particular item is hard to predict, making it risky to encode selectively. Understanding how the memory system responds to these constraints is important for many reasons. For example,

episodic details acquired during task performance eventually become the semantic structures of expertise, so understanding the structure of episodic memory and how it evolves should help us improve how experts are trained. Such an understanding would also be a basis for memory-friendly designs of tools like programming environments.

Studying memory in expert problem solving is difficult precisely because the task environment is complex and requires the problem solver to process a large amount of information. This complexity makes it impractical to discover the contents of memory experimentally—indeed, from the time of the nonsense syllable (Ebbinghaus, 1885/1964) experimenters have attempted to isolate memory from the influence of knowledge (arguably to the detriment of theory; Young & Lewis, 1999). And although other common analytical methods are not burdened by the need for experimental control, these have their own problems. For example, task analysis (e.g. Kirwan & Ainsworth, 1992) attempts to specify the structure of the task environment, and is often augmented with naturalistic observation (Hutchins, 1995a; Zsambok & Klein, 1997) or interviews to elicit what operators (think they) do as they perform. These approaches produce useful insights, but as measurement tools they are not sensitive enough to reveal the underlying memory processes with any precision. A more detailed form of observation is microgenetic analysis (Siegler, 1991), which attempts to track cognitive behavior at a fine grain, often across multiple trials on the same task to allow inferences about what is being learned. However, the very richness of detail that places episodic memory at the center of expert problem solving also places an overwhelming bookkeeping burden on the analyst using any of these approaches to model behavior.

The challenge, then, is to study episodic memory in expert problem solving with both precision and completeness, coping with the fact that these two requirements interact to multiply the cost of the analysis. Precision entails a detailed representation of what a problem solver is thinking moment by moment, and completeness requires that this detailed representation achieve broad coverage of the information being processed. This coverage is critical because the validity of model is jeopardized by every item of information processed by the problem solver but not represented in the model.

One approach to achieving precision and completeness concurrently is to simulate behavior computationally. Simulation adds precision because the knowledge attributed to the problem solver, and the information flow between problem solver and environment, is represented as executable code sufficient to reproduce the target behavior. Simulation supports completeness because it solves a critical aspect of the bookkeeping problem: any change made to a computational model, for example the addition of new knowledge, can be tested by running the model and comparing its output to the data and to the output of other versions of the model.

Beyond supporting precision and completeness, computational simulation is perhaps the only reliable vehicle for bringing cognitive theory to bear on complex behavior. With the advent of programmable cognitive theories (Newell, 1990; Anderson & Lebiere, 1998), the analyst can not only simulate behavior, but can subject the simulation to the same cognitive constraints faced by humans. Such constraints, for example limitations on the capacity and span of working memory, help to ensure that the model is a valid representation of cognition. More importantly, and as I illustrate in this article, theoretical constraints on a model of behavior generate novel insights into the underlying cognitive structures and processes.

Applied to expert problem solving, cognitive simulation produces sufficient insight into human memory to warrant naming a new construct. The construct of *near-term memory* combines the analytical goals of long-term working memory (Ericsson & Kintsch, 1995; Ericsson & Delaney, 1999) and other non-computational approaches to studying real-world memory (Yntema, 1963; Neisser, 1976; Venturino, 1997) with the methodological and theoretical leverage of cognitive simulation. The moniker "near-term" derives from the structure of the information stored, which combines attributes of short- and long-term memory. Short-term memory as typically studied in psychology is largely episodic, in that it maintains items in context (for example, the fact that the syllable BAZ appeared in the list of words given at study time). In contrast, long-term memory is typically analysed for its organization and semantic content, as illustrated by the standard approaches to expertise mentioned above. Near-term memory bridges the gap, storing large quantities of episodic detail and linking it to semantic knowledge in such a way that critical items can be retrieved if the need arises and the cues are available.

Near-term memory is bound by two constraints flowing from the nature of the task interacting with the nature of cognition. The first constraint is that problem solving in a complex, information-rich environment demands relatively non-selective storage of information in memory. When someone is truly searching a problem space, essentially any item of information could turn out to be important later and therefore beneficial to recall at the right time. In contrast, if the problem solver knew ahead of time what information would become relevant later, arguably he or she would not be searching a problem space as much as exhibiting skilled or over-learned behavior. One consequence of the non-selectivity constraint is that near-term memory stores structurally diverse kinds of information, including perceptual chunks, recoded semantic knowledge, and episodic pointers into the environment. A second consequence is that encoding is a continual process, because there is continual processing of task-specific information and because essentially any item processed now could be important to remember later. The general implication is that near-term memory contains vast amounts of information, much of which is never retrieved, but any of which could be retrieved and could critically affect the course of behavior.

The second constraint on near-term memory concerns the actual circumstances of retrieval. In a complex task that involves problem-solving search or a dynamic environment, an item could become relevant essentially at any time. Thus, the lag between encoding and retrieval can be many minutes long. Moreover, there is no guarantee that maintenance or elaboration will be possible in the interim, because the retention interval will generally be filled with other cognitive activity. The consequence is that encoding specificity (Tulving, 1983) is one of the main factors governing the availability of a memory item. The encoding specificity principle says that when an item is encoded, other items of working memory are encoded with it, and will serve as cues for the item later. If there is time to elaborate an item, the set of cues encoded with it grows and retrieval will be easier. However, expert problem solving generally involves concentrated or event-filled activity that affords little slack time in which to elaborate an item, so the item's cues will remain limited. Should the right cue enter working memory in the future, for example if the environment is dynamic and the cue simply appears, then the target item will seem to pop to mind. The general implication is that expert problem solving

depends critically on cues that retrieve details about the task, and hence on the semantic knowledge and perceptual information of which these cues are comprised.

Near-term memory, then, is memory of the task that expert problem solvers (in this study a programmer) build up as they perform the task. The main functional requirement on near-term memory is that it captures as many of the intermediate products of problem solving as possible for use later, because any particular detail could become relevant again. Measurement and analysis of near-term memory are performed on a computational cognitive model. Because this model is strongly constrained by theory and data, and because it must run, it fills in details that cannot be discovered through experimentation or simple observation.

The rest of the article fleshes out these ideas and their implications. Section 2 describes the programming episode that furnishes the behavioral data. Section 3 describes a computational model of this episode and constraints on the model's learning and memory processes. Section 4 analyses the model's near-term memory quantitatively and structurally. Section 5 assesses the model's validity in representing the target behavior. Finally, Section 6 draws implications for expert-novice differences in programming and inspects the model to reveal low-level but pervasive cognitive costs of environmental clutter.

2. Task and behavioral data

The programming session from which the sample behavior was drawn was one session in a long-term, multi-person project to create a large natural-language comprehension system (this system will be referred to simply as the *program*). The programmer contributed regularly to implementation, and was thoroughly familiar with the theory behind the program and the production-system language in which the program was implemented. Her high-level goal for the session, during which she worked alone, was to make a specific change to the implementation. The session lasted 80 min, during which the programmer stepped through a single run of the program interactively at a fine grain. The interaction took place in a GNU Emacs process buffer, a virtual teletype with a prompt at the bottom at which a user can communicate commands to a running process. The process with which the programmer interacted was a production-system interpreter into which the program had been loaded. In addition to issuing stepping commands to advance the program, the programmer issued query commands to request various kinds of state information from the interpreter, including the contents of data structures, the contents of the runtime stack, and elements of code.

All information printed by the language interpreter appeared at the bottom of the process buffer, with Emacs automatically scrolling old output off the top of the screen when more room was needed at the bottom. The old output remained accessible, but had to be scrolled back into view using keyboard commands. Thus, the screen was functionally a window onto the process buffer, with the default window location being over the command prompt at the bottom, but with window location adjustable through scrolling commands. Emacs had been instrumented for this study to record the contents of the process buffer, as well as a time-stamped keystroke protocol. The programmer thought aloud, and her utterances and gestures and the contents of her display were recorded on videotape.

From the 80-min session a 10.5 min interval was selected for detailed analysis through computational simulation. This interval was chosen for its relatively high concentration of navigation to hidden information, the focus of an in-depth analysis of memory for external information (Altmann, 1996; Altmann & John, 1999). Of the 26 scrolling episodes during the 80-min session, five occurred in the modeled interval. The amount of text scrolled in the 26 episodes totaled 2482 lines under a window of 60 lines, or roughly 41 windows-full. This extensive scrolling underscores the importance of information access in the large information spaces of many real-world tasks.

3. A model of the behavioral data

This section describes the salient aspects of a process model of the programming behavior described above, in preparation for examining the model's near-term memory. The simulation—the model as it runs—reproduces the programmer's verbal and key-stroke protocols at a fine-grained level; aggregate descriptive statistics on the model's function are given in Section 5. Here the model will be described in high-level functional terms. The first subsection characterizes the model's performance in terms of its main goals, processes, and actions. The following two subsections describe the other knowledge the model brings to its task, including semantic knowledge representing expertise in the domain of programming and episodic knowledge that arises in the course of task performance. Finally, the last subsection describes how the protocol data and the theoretical assumptions represented in Soar interact to shape what and how the model learns.

3.1. PERFORMANCE ASSUMPTIONS

The model's main mode of performance is a kind of comprehension—it tries to gather information about items in its environment. This is a simplified representation of the programmer's primary activity during the modeled interval, in which she is reminding herself how the program works by stepping through it in detail. The model does not construct the complex mental structures associated with comprehension of text in general (e.g. Lewis, 1993; Kintsch, 1998) and programs in particular (Brooks, 1983; Green, Bellamy & Parker, 1987; Pennington, 1987; Wiedenbeck, 1991; Von Mayrhauser & Vans, 1996). Rather, the focus is on what low-level knowledge is acquired on-line as a side effect of performance, a question that has received much less attention.

The model selects goals to comprehend program objects, for example a code fragment or a data structure. The model also issues commands to change the display. Some commands generate new information and some scroll to old information. The model uses this external information as it tries to comprehend objects.

To comprehend an object, the model retrieves information about that object, either from the screen (an external source) or from LTM (an internal source). For example, suppose the model is reading a story about animals and the screen is displaying the word CAT. [This reading task is fictitious, but used for clarity. The domain the programmer was working in is described in some detail in Altmann (1996) and Altmann & John, (1999)]. The model would attend to CAT, adding a semantic representation to working

memory (WM).

Item CAT   The result of attending to CAT.

Alternatively, if CAT were not on the screen but the model had semantic knowledge about cats, it could produce the element CAT in WM from LTM. The purpose of doing this might be to probe its semantic knowledge for information about cats, using CAT as a cue. Probing and attention produce identical structures in WM.

Item CAT   A probe on CAT.

Probing is also a way to retrieve episodic information about what was attended in the past. For example, if the model attended to the word CAT, an episodic record of this attention event would be stored in LTM. (The learning mechanism that creates this episodic record is described later, in Section 3.4.) If the model later decided to probe its memory with CAT ("Didn't I see the word CAT?"), it would recall having attended to CAT at some time in the past ("Yes, I believe I did."). An episodic trace for having attended to something is one of several kinds of information stored in near-term memory.

The main functional role of the probing process is in fact to place cues in WM to retrieve related knowledge. Such retrieval is one basic process of comprehension, a prerequisite to building up the mental structures that represent the current situation (Kintsch, 1998). Success in building these structures thus depends on the right retrieval cues entering WM at the right time. The more semantic knowledge it has from which to generate good cues, the better the model can comprehend the situation. This semantic knowledge, viewed as a comprehensive retrieval structure, is considered one of the main components of expertise (Ericsson & Kintsch, 1995). Later sections will argue later that this is only half the story, but the nature of this semantic knowledge is examined next.

3.2. SEMANTIC KNOWLEDGE

The model contains semantic knowledge of the kind we would expect a skilled programmer to bring to a programming task. This knowledge is static in the sense that it is loaded before the simulation begins and persists in the same state throughout. It consists of knowledge about the particular program to be modified, including its structures and processes; about the implementation language, including its central concepts and idioms; and about computer science fundamentals, like data structures and algorithms. Such knowledge is typically found in expert systems and other symbolic AI programs. The model also knows all it needs to know to use the programming environment.

Semantic knowledge serves several purposes in the model, but the one most important for near-term memory is that knowledge is a source of retrieval cues. Specifically, semantic knowledge is the source of probe items placed in WM. For example, in the fictitious reading task introduced above, the story might involve cats and birds, and the model might know enough to probe its memory to see whether one represents a threat to the other.

Another category of semantic knowledge consists of skills that have become essentially automatic. One such skill is serial attention, which directs the train of thought using a combination of knowledge in memory and cues in the environment (Altmann, 1996).

A second skill is to prefer to attend to new objects; this heuristic automatically directs the model to examine the most recent information to appear on the display. Such skills are more basic and more general than what we commonly think of as expertise, but are nonetheless essential components of performance knowledge. The importance of such skills in performing a given task does not become obvious until one tries to simulate performance computationally, at which point it becomes clear that the model cannot perform without them. The power of simulation to reveal cognitive functionality was anticipated in Newell's early call to develop "complete processing models" (Newell, 1973) and remains one of the benefits of simulation obtainable in no other way (John & Altmann, 1999).

3.3. EPISODIC KNOWLEDGE

If one-half of expertise is stable, long-term semantic knowledge, the other half is situation-specific episodic knowledge. For example, an object may appear on the screen, perhaps in response to a query to the language interpreter, and then may disappear again as other output displaces it. If the model is to retain any kind of memory for this object, cognition must generate a new symbol in memory and associate it with the item that came and went. In the model, a stream of internal episodic symbols or event tags is produced automatically by the cognitive system. Every few seconds (specifically, every time a new comprehension goal is set), the current tag is replaced with a new tag. Then, when the attention process adds some new item to WM in service of the current comprehension goal, it automatically associates the current event tag with the attended item. This act of association causes Soar's learning mechanism to link the tag and the item together in memory, forming an episodic trace whose cue is the item itself.

3.4. CONSTRAINTS ON THE MODEL

The programmer in the study clearly remembered much of what had appeared on the display; otherwise she could not have known to scroll to the information she needed. On the other hand, she clearly did not remember everything about what had appeared on the display, or she would never have needed to scroll (she would have already known everything). Moreover, hundreds of objects appeared on the display during the programming session, but she only referred back to a few of them. The need to refer back to any particular object was contingent on events that would have been difficult to anticipate when she first attended to the object. Therefore, it is unlikely that she deliberately stored memories of only the few objects she would eventually need. Converging evidence for this conclusion is that the verbal protocol contains no evidence of deliberate storage; that is, there are no comments like, "Oh, I better remember that!," which one would have expected to see from time to time in the protocol if memory for objects depended on explicit intentions represented in WM (Ericsson & Simon, 1993). These data represent constraints on what the model must and must not learn if it is to be an accurate model of the programmer's behavior. Specifically, the model must learn a little about the many objects that the programmer attended, with relatively little effort invested per object. On the other hand, the model must not learn so much about each object that scrolling would be unnecessary.

Soar specifies how the model must meet these constraints imposed by the data. The primary source of architectural constraint is Soar's associative-learning algorithm, chunking, which encodes new productions or chunks in LTM. (Productions, or simple IF–THEN rules, are the elemental form of knowledge in Soar.) To encode a chunk, Soar collects relevant elements that were in WM as a problem-solving episode began and links them to the result of that episode. The resulting chunk allows the problem-solving episode to be skipped next time the relevant elements appear in WM. In place of the episode, the chunk simply deposits the result in WM. When this happens we say that the chunk fired or, equivalently, that the chunk was retrieved from memory.

Chunking accounts for a broad range of learning phenomena (Newell, 1990) in part because it specifies how they must be represented. The key constraint is that chunking implies a need for deliberative processing to recall an item. This contrasts with how Soar learns to recognize an item, which takes little effort.

For example, suppose Soar's task were to study the stimulus CAT → DOG and to recall DOG when tested with the stimulus CAT → ?. At study time, the image CAT → DOG is in WM and is relevant to the task of learning that CAT should evoke DOG. Therefore, Soar builds a chunk that links DOG to the image CAT → DOG.

IF   CAT → DOG is in WM,
THEN   answer DOG.

This new chunk recognizes the stimulus CAT → DOG, and responds DOG. However, CAT → DOG is different from CAT → ?, so the new chunk will not recognize CAT → ? at test time and therefore will not serve to recall DOG. This strict limitation on what cues will enable a chunk to fire represents an implementation of encoding specificity (e.g. Tulving, 1983).

Prior Soar models have demonstrated a mechanism called *data chunking* that uses deliberative processing at test time to compensate for encoding specificity at study time (e.g. Newell, 1990). Data chunking employs a generate-and-test process to break the link between an external stimulus and a desired response. In the example above, data chunking would break the link between CAT → DOG and DOG at test time, by using knowledge stored in memory to generate candidate responses to CAT → ?. For example, Soar might generate CAT → BIRD, CAT → MOUSE, etc., until it happened on CAT → DOG. At this point the chunk above would fire, recognizing that DOG was what had been associated with CAT in the original stimulus.†

A unique theoretical premise of Soar is that the cognitive system learns associatively all the time. Put another way, the assumption is that cognition is always producing new information, and memory is always storing this new information to prevent having to redo the same work in the future. Stored knowledge is not perfectly accessible, though,

---

† In many Soar models the next and final step of the data chunking process would be to capture the response DOG in a new chunk.

IF   CAT → ? is in WM,
THEN answer DOG.

This new chunk is linked to the stimulus CAT → ? instead of the stimulus CAT → DOG, because CAT → ? was the image in WM when the generate-and-test process began. In future, the new chunk would recall DOG when cued with CAT → ?, which is the desired behavior. The step of learning this new chunk is necessary when the task involves free recall, namely recall in the absence of the stimulus from which the item was learned.

because the chunking mechanism has essentially to guess heuristically which cues to associate with an item. The algorithm for selecting cues (called backtracing; Laird, Rosenbloom, & Newell, 1986; Howes & Young, 1997) has to be conservative to spare cognition from being continuously inundated with irrelevant information, and the result of this conservatism is encoding specificity. Nevertheless, the completeness of coverage achieved by the chunking mechanism, by virtue of being "on" all the time, helps to satisfy the non-selectivity constraint identified earlier. That is, continual learning provides the functional benefit of storing episodic detail in near-term memory with broad coverage and little selective exclusion.

As a consequence of continual learning, the model learns three kinds of chunks as the contents of near-term memory. The quantity, structure and functional role of these chunks is discussed next.

4. The model's near-term memory

The perfect problem-solving system might store in near-term memory every detail it ever attended to, linked to cues such that an item would be recalled if and only if it were relevant. Such memory would make problem solving quite efficient, because work would never have to be redone. The argument here is that expert problem solvers are closer to this ideal than other people, by virtue of the contents of their near-term memory. This section analyses the quantity, structure, and functional role of the chunks encoded by the model as it runs, taking these to be a representation of the programmer's near-term memory.

4.1. QUANTITATIVE ASPECTS

The individual steps of the programmer's behavior at the second-by-second level, and the corresponding steps in the simulation, are too numerous to analyse without some kind of aggregation. For example, the combined number of comprehension goals, attention events, and probe events in the simulation totals 499 steps, or roughly one step per second. Below the level of these steps, the model fires tens of productions per step (as quantified below). Because of this quantity of detail, the focus here is on higher-level analyses of the inputs to and outputs from near-term memory rather than a step-by-step analysis of the simulation itself.

An inventory of the model's knowledge and its frequency of use is shown in Table 1. Each element of knowledge in Soar is represented as a production, and each cell in Table 1 presents a total number of productions or production firings. When the simulation has completed, long-term memory contains 1514 productions in total, as indicated in the Production Count column. Of these, 194 were pre-loaded, representing the semantic knowledge and skills described above. The remaining 1320 are chunks encoded during the simulation, representing near-term memory. The Firing Count column shows how often productions in various categories fire during execution. There are 17 352 firings in total, of which 15 851 (91%) are from pre-loaded productions and 1501 (9%) are from encoded productions. In terms of real time, this translates to about 30 firings per second, with roughly three of these due to retrievals from near-term memory. Finally, the Number Fired column shows how many productions in each category fired at least once

TABLE 1
Inventory of knowledge in the model

	Production count	Firing count	Number fired
Pre-loaded knowledge			
Expert knowledge	126	2848	126
Automatic skills	68	13003	68
Total	194	15851 (91%)	194
Near-term memory			
Perceptual chunks	407	354	152
Semantic chunks	257	62	44
Event chunks	462	951	262
Other	194	134	33
Total	1320	1501 (9%)	491
Total			
	1514	17352 (100%)	685

Note. Each cell specifies the number of productions or production firings for that category. Productions that were *pre-loaded* were in memory when the simulation began, and productions in *near-term memory* were encoded as the simulation proceeded. The *production count* for near-term memory is the number of productions encoded during the simulation, the *firing count* is the total number of production firings for that category, and the *number fired* is the number of productions in that category that fired at least once.

during the simulation. For pre-loaded productions this is equal to the production count, but for near-term memory productions the number fired varies, reflecting the fact that some productions never fire after they are encoded. As percent of production count, the number fired ranges from 17% for semantic chunks (44/257) to 57% for event chunks (262/462). Thus, in each category a substantial number of new productions are triggered sometime after they are encoded. The implication for human near-term memory is that details stored moment-by-moment during problem solving influence the course of behavior in an on-line, feed-forward manner.

4.2. STRUCTURAL ASPECTS

This section examines the structure of chunks in each category shown in Table 1, and evaluates their functionality and psychological plausibility. The category labeled "other" is not analysed; the productions in this category are difficult to interpret and are best considered artifacts of the model's representation interacting with Soar's universal learning.

4.2.1. *Event chunks*

When the model attends to an item on the display (e.g. CAT), Soar creates an event chunk in memory that maps the semantics of the item to episodic information indicating that the item was attended (Altmann & John, 1999). The general form of this chunk is

    IF   an item is attended and placed in WM,
    THEN   add an event tag to WM indicating that the item was attended.

The existence of such a production represents a memory for having attended to an item.

Encountering the same item again will cause an event chunk like this to fire. This firing retrieves the event tag, which itself supports a number of inferences. For example, one inference is that the item exists in the environment. Based on this inference, the model can decide whether or not to pursue the item externally by scrolling to it (Altmann, Larkin & John, 1995; Altmann & John, 1999). A second possible inference is that other, untagged items in WM are novel in that they have just now been attended for the first time. This allows for the selection of novel items over old ones, a heuristic used by the programmer from time to time (Altmann, 1996).

A more general capacity for episodic memory would allow recall of much more complex information. Evidence for such a general capacity in the programming domain comes from a study of software designers (Jeffries *et al.*, 1981). Expert designers used episodic memory to recall pending design questions, where each question comprised a complex network of interrelated memory structures. Similarly, the problem-solving method of progressive deepening (Newell & Simon, 1972) involves recreating previous cognitive states for the purpose of integrating new information. For example, progressive deepening in algorithm design (Kant & Newell, 1984) involves mentally simulating a computation to determine where and how it is under-specified, then "resetting" the mental run with the new information integrated into the mental representation of the program. Thus, in general, people use near-term memory to store complex but coherent structures that are jointly episodic and semantic.

4.2.2. Perceptual chunks

The model acquires a kind of low-level perceptual chunk (Chase & Simon, 1973) that speeds up attention once it is learned, and through this efficiency improves overall comprehension. Each perceptual chunk is a new production that maps a feature's external representation, plus various internal cues in WM, to that feature's internal representation. The general form of this kind of production is

    IF   the goal is to comprehend something and
            there is a feature on display and
            cues in WM prompted attention to the feature,
    THEN   add that feature to WM.

The internal cues indicated in the conditions are the WM elements that caused the model to attend to the feature in the first place, including the comprehension goal selected at the time. These conditions are included because (as discussed earlier) Soar's chunking mechanism is conservative in its inferences about what caused a result. In this case, Soar assumes that any WM element processed in service of the current goal up to the time the feature entered WM shares credit for that feature entering WM.

Once a perceptual chunk is created for a given feature, then if the feature appears on the display again, with the same elements in WM, the chunk will directly place the internal representation of the feature into WM. The savings due to a perceptual chunk thus lie in compiling out the cognitive bottleneck of attention (Schneider & Shiffrin, 1977).

This caching of features in recognitional memory plays a key role in the model's ability to comprehend objects. Objects in the environment often have a hierarchical structure.

For example, data structures in a program typically have fields. When the model tries to comprehend a particular data structure, it knows it should look at the data structure's individual fields. However, the model may not get through all fields before selecting a new comprehension goal, say if it pursues a depth-first path to understanding a particular field in detail before continuing with the others. If the model is interrupted in the middle of a data structure, information about the fields that did get through is cached in perceptual chunks. These chunks will fire immediately if the model returns to the goal of comprehending that data structure. This in turn lets the model get past the fields it attended before and attend to new ones.

Perceptual chunks are not only functional but plausible. The incremental comprehension they produce is one way people learn about their environment (Rieman, Young & Howes, 1996). More generally, the learning and deployment of display-based rules is a common assumption in many accounts of interactive behavior (Larkin, 1989; Rieman, Lewis, Young & Polson, 1994; Gray, in press) and expertise (Chase & Simon, 1973). Soar does, however, limit the functionality of perceptual chunks by tying their conditions strictly to sensory cues. That is, for a perceptual chunk to fire, it must be activated by information impinging on the senses rather than images generated from memory. This makes perceptual chunks display-based in the most literal sense, in that they contribute to the ability to recognize but not to the more powerful ability to recall.

This recognitional capacity is itself subject to the attentional selection that humans exhibit (see, for example, Pashler, 1997). Because the goal is one of the internal cues that prompts the model to attend to something, perceptual chunks include the goal as a condition. This prevents over-learning of the display to the point where every feature enters WM in parallel as soon as the display contents reappear. Thus the model, as a function of the underlying learning theory, maintains goal-based control over the influx of information from the display, even after learning.

To summarize perceptual chunks contribute a limited efficiency improvement to the retrieval of information. This improvement is critical, however, because it saves the model from having to start accumulating information from scratch whenever it selects a given goal.

4.2.3. Semantic chunks

Psychological accounts of semantic memory are traditionally cast in terms of networks (e.g. Anderson & Bower, 1973) and the current model is no exception. Productions can be interpreted as links between one semantic node on the condition side and another on the action side. On this view, the learning of new productions represents the addition of new links to the semantic network. The result of each new link to a given concept is that the knowledge related to that concept becomes more directly related and hence more quickly accessed. This restructuring enables the rapid access to long-term semantic knowledge that helps distinguish experts from novices (Ericsson & Kintsch, 1995).

The general form of semantic chunks in the model is:

IF   the goal is to comprehend something and
           cues in WM were used to probe memory,
THEN   add the retrieved fact to WM.

This is similar to the perceptual chunks discussed earlier. The internal cues are again the WM elements that caused the model to recall the fact in the first place, including the

comprehension goal selected at the time. If these elements appear again in WM, the recoded fact will be placed in WM directly. The step of explicitly probing memory in service of the comprehension goal has been compiled out.

Like perceptual chunks, semantic chunks play a key role in the model's ability to comprehend objects, and for essentially the same reasons. As comprehension processes revisit a particular mental state, that state expands to include more information as more facts come within a single retrieval of one another. The prediction is a relatively gradual and consistent improvement in the efficiency of retrieving related clusters of knowledge. This prediction is consistent with power-law learning of individual problem solving strategies (Delaney, Reder, Staszewski & Ritter, 1998), and is also consistent with programmers becoming faster and more accurate at identifying important elements of code as their experience increases (Davies, 1994).

Semantic chunks by themselves cannot fully account for the acquisition of semantic knowledge, which depends on subsequent generalization processes. For example, the production above includes conditions that test for the presence of other cues in WM beyond just the object being comprehended. Soar predicts that to generalize the knowledge contained in such a production, cognition must engage in a deliberate, reflective process that produces a new production without the restrictive conditions (this is the data chunking process referred to above; Newell, 1990; Lehman, Laird & Rosenbloom, 1998). Thus, the model embodies the hypothesis that automatic recoding of semantic knowledge during task performance is only part of the transition from novice to expert. As with deliberate practice (Ericsson & Lehmann, 1996), a subsequent phase of deliberate, inferential processing on the contents of near-term memory is necessary to transform them into the fully general semantic structures of expertise.

4.3. DISCUSSION OF NEAR-TERM MEMORY

The analysis above provides a preliminary characterization of near-term memory along both structural and quantitative lines. Structurally, near-term memory has the following characteristics. First, it is diverse in being perceptual, episodic and semantic, arguably running the gamut of declarative knowledge. Second, near-term memory is context-specific. As a function of the underlying cognitive theory (Soar), sweeping storage of cognitive events is achieved for the price of restricted on-line generalization. Third, despite this specificity, near-term memory is nonetheless functional. Perceptual and semantic chunks both increase the amount of knowledge that can be accessed by a single retrieval, supporting behavior like incremental comprehension. Similarly, event chunks are functional in that they serve as an index or inventory of objects existing in the environment.

Quantitatively, the measurements above suggest that near-term memory is sampled far less often than semantic knowledge brought to the task. This is reflected in Table 1 in the proportion of firing counts to productions for near-term memory (1501 firings, 1320 productions) compared to that proportion for pre-loaded productions (15 851 firings, 194 productions). This imbalance reflects the assumption that near-term memory involves large amounts of encoding but little generalization, whereas semantic knowledge is explicitly generalized to transfer to other contexts. Given this assumption, though, the noteworthy finding is that near-term memory is accessed as often as it is. From a functional perspective, the model's behavior is critically affected by retrieval from

near-term memory. The clearest example is an event chunk firing, which leads the model to embark on an excursion through information space.

A second quantitative assessment is that episodic knowledge about the environment (perceptual and event chunks) is a much larger component of near-term memory than recoded semantic knowledge. The ratio of environment-related chunks to semantic chunks is 3 : 1 in terms of productions encoded and 21 : 1 in terms of productions fired (Table 1). This must to some extent reflect the representational choices made in building the model. Were the model to construct more complete mental structures representing the situation, pervasive recoding would occur during those processes as well, giving a more accurate picture of the balance between the various categories of near-term memory. However, as the model stands we can still infer that detail describing the environment is stored in large quantities and is sampled often enough to influence the course of problem solving. This is in general agreement with the situated view of cognition (e.g. Suchman, 1987), and with the view that visual attention serves largely to bind variables as needed by cognitive programs (Ballard, Hayhoe, Pook & Rao, 1997).

5. Model validity

A central claim of this article is that one can inspect a computational cognitive model, as was done above, to make inferences about human cognition, in particular near-term memory. For such inferences to be valid, the model must conform to whatever constraints are available from theory and data. When a model is implemented in a cognitive architecture like Soar, it is shaped by theory automatically. However, the model then has to be compared to the behavioral data to see if it conforms to that as well (and changed if it does not).

The validity of the model is tested here in two ways. The first test is to compare the steps taken by the model during the simulation to the steps taken by the programmer. This comparison is made in terms of keystrokes but not formally in terms of verbal utterances, for reasons discussed below. The second test evaluates the grain size of cognitive events represented by the model.

5.1. FIT TO PROTOCOL DATA

Though the verbal protocol provided critical guidance in building the model, the formal measure used in the model-fitting process was the model's reproduction of the programmer's keystroke protocol. Keystrokes are relatively unambiguous to code, and occurred every few seconds on average and thus provided a reasonably fine-grain assessment of how well the model's train of thought matched the programmer's.

The keystroke protocol was segmented into 50 commands, where a command might be an instruction to the programming environment to scroll one screen worth of information. Consecutive commands related to the same purpose, for example a sequence of scrolling commands intended to redisplay a particular hidden item, were reduced into a single command. Nineteen of the original 50 were reduced to seven by this procedure, leaving a total of 38 target commands in the data set (31 + 7). The model itself issues 34 commands. These map, in correct sequential order, to 34 of the 38 target

commands. The four target commands that the model fails to generate involve display-manipulation knowledge omitted from the model's simplified representation of its environment (Altmann & John, 1999). Thus, 34 of 38 commands, or 89% of the target data, are successfully generated during the simulation.

5.2. GRAINSIZE OF EVENTS

Beyond the tracking of physical actions, it is also possible to analyse the temporal granularity of the model for insight into the extent to which it was constrained by data. The simulation spans 629 s, in which it takes 499 steps. (A step consists of setting a comprehension goal, attending to an object, or probing memory with a cue.) Mean time per step is thus 1.3 s. The 1-s level is about the lowest at which one expects to find useful evidence in protocol data (Ericsson & Simon, 1993). Also, evidence from at least two domains suggests that encoding a goal or intention takes at least a second, even for short-term goals that are only active a few seconds at a time (Altmann & Gray, 1999; Altmann & Trafton, 1999). That the model operates at this level is evidence that it reaches the limits of constraint available from the data.

6. General discussion

This article introduced the construct of near-term memory to describe naturalistic memory processes in precise computational terms. The defining characteristics of near-term memory are (1) a large volume of candidate to-be-remembered items and (2) unpredictable need for any particular one. Determining what information is actually stored in and retrieved from memory under such circumstances is difficult because the circumstances preclude experimental control. Instead of analysing the contents of memory directly, the approach illustrated here involves conducting an extended, detailed computational simulation based on the dual constraints of protocol data and cognitive theory. The model is then inspected to describe the structure of near-term memory and the rate at which its contents are stored and retrieved. The central role of cognitive simulation distinguishes the construct of near-term memory from long-term working memory (Ericsson & Kintsch, 1995), distributed representation (Zhang & Norman, 1994), situated action (Suchman, 1987) and other approaches (Hutchins, 1995b) to analysing memory in natural settings.

6.1. PSYCHOLOGY OF PROGRAMMING

What does this study tell us about how expert programmers think? The primary contribution is a new perspective on the role of expert knowledge. One typically thinks of expertise in terms of its usefulness in interpreting what things *mean*. For example, programmers have the expertise to know what the code means that they are looking at. A more general example is that literate adults have the expertise to comprehend the language they are reading (Kintsch, 1998).

However, the model discussed here shows that expertise plays an equally important role by *mediating access* to things. Event chunks provide the clearest example of this

mediation. The programmer scrolls a number of times during the modeled interval—that is, she shifts her window to a new location in the information space. In each case, she had a clear goal for the scrolling action. The protocol data indicate that she knew exactly what object she was looking for and roughly where it was. Therefore, some kind of memory reminded her of a specific object in the information space that she had seen before, that was now hidden, and that would be useful to go inspect. What were the contents of this memory? And what cued their retrieval?

In the model, this memory consists of an event chunk linking the target item with an event tag. Retrieving this memory is the end result of a chain of retrievals through semantic memory. The chain begins with an item in the environment reminding the model of something, which triggers more remindings, which eventually (a few seconds later) retrieve the target item, which finally retrieves the event tag. The event tag reminds the model that it saw the target on the display. At this point the model infers that a trip through information space to visit the target would be worthwhile (Altmann & John, 1999), for which the overt evidence is a scrolling command.

This chain of events seems remarkably complex for such a simple action as scrolling. However, the model suggests that this complexity marks the difference between a novice and an expert. The novice is more likely to have breaks in the chain that prevent the critical final reminding that the item is there in the environment to be inspected. The expert, in contrast, understands that a particular item on display is an important cue, is reminded of items relevant to that cue, recalls having seen one of those items and finally knows to scroll to inspect that item. In short, the expert learns more easily where things are and knows when and where to look for them, in addition to knowing what they mean when he or she finds them.

6.2. PSYCHOLOGY OF MEMORY

Can cognitive psychology benefit from yet another memory construct? I believe it can, based on recent developments in research on cognitive architectures. Controlled experimentation is not a viable way to investigate naturalistic memory, and the large-volume, detailed approach to process modeling illustrated here is too labor intensive and error prone to be feasible with traditional hand-simulation methods of cognitive task analysis (e.g. Gray, John & Atwood, 1993; John & Kieras, 1996). For these reasons, some naturalistic memory processes have simply not been targets of psychological research. Advocated early in the cognitive revolution (Newell, 1973), cognitive architectures have only recently evolved to the point where the detailed study of memory in everyday settings is possible. As a result, the number of process models that acquire new representations on-line and whose performance has been compared to human behavior is still quite small (Ritter & Bibby, 1997). However, improvements in computational efficiency now allow the construction of models with millions of productions (Doorenbos, 1995) and improvements in accessibility have analysts publishing executable models for inspection and for use as building blocks by other researchers (Anderson & Lebiere, 1998). These developments offer an opportunity to refocus on behavior that psychology has ignored to date for lack of tools and methodology. We are now in a position to analyse memory in expert problem solving, where information is voluminous and complex and where any item could be important to remember later.

The closest existing construct to near-term memory is long-term working memory (LT-WM; Ericsson & Kintsch, 1995; Ericsson & Delaney, 1999). The two constructs are closely related in targeting naturalistic memory. However, LT-WM supports essentially routine performance on complex tasks; performance is made routine by extensive domain knowledge that allows the expert performer to anticipate retrieval demands and encode items selectively and with attention to resisting interference in repetitive tasks. In contrast, near-term memory supports problem solving search, in which anticipating retrieval demands is precisely the difficulty. Here the primary constraint is coverage—without knowing what the critical object will be, it pays to encode a little about each object one attends to. However, to avoid trivializing this constraint, near-term memory needs to be operationalized in terms of sound limits on the power of the encoding process and in terms of realistically high volumes of information in the task environment. Thus, near-term memory is particularly suited to simulation within a cognitive architecture that contains a theory of learning and that is computationally efficient.

6.3. HUMAN-COMPUTER INTERACTION

What can near-term memory tell us about how people interact with computers? Because the analysis is grounded in a computational cognitive model, it furnishes a precise representation of the underlying encoding and retrieval processes against which to evaluate the consequences of interface structure. This section discusses two such consequences. The first concerns responses at the memory level to clutter in the environment, and the second examines the potential for browsing indicated by the model's encoding and retrieval processes.

6.3.1. The cost of clutter

The model's spatial knowledge is limited to distinguishing what is hidden from what is visible (Altmann & John, 1999). This simplifies away most of the complexity inherent in interfaces to large information spaces. Suppose, then, that the model's knowledge of the world were elaborate enough to represent multiple locations for features, such as different buffers or windows. How would the model know where to look when it wanted to find something?

A well-structured interface would facilitate the cognitive aspects of search for hidden objects by offering a consistent and comprehensive mapping from objects to locations. If the model knew and could apply this mapping, this would represent survey knowledge (Golledge, 1991) allowing for orientation within the information space. For example, programming environments often deposit different kinds of output into different windows. Good survey knowledge of a well-structured programming environment would make it easy to infer the containing window, or the direction in which to navigate, from the to-be-located object. Acquiring such knowledge would take time, however, introducing a penalty for novice users, and learning time might increase with complexity of the object-location mapping.

By contrast, the experienced user of an ill-structured interface might employ a strategy of encoding location information, as the model now encodes event chunks. Location

information could then be retrieved directly from memory, rather than having to be inferred as needed. However, to make use of such location chunks, the model would need to probe with location cues as it now probes with items. That is, the model would have to imagine candidate locations to ask itself where it might have seen the item of interest. This kind of generate-and-recognize process (Kintsch, 1970; Anderson & Bower, 1972; Watkins & Gardiner, 1979) is deliberate or controlled and therefore entails a cognitive opportunity cost. Millisecond differences can systematically affect the choice of interaction strategy (Olson & Nilsen, 1988; Lohse & Johnson, 1996; Ballard *et al.*, 1997; Gray & Boehm-Davis, in press) and even a small overhead to generating location cues may pervasively discourage trips through information space. In addition, the chance of being interrupted by an intruding goal increases with time spent maintaining attention on the current goal (Altmann, 1996; Altmann & Trafton, 1999).

The model thus allows informed speculation about how clutter translates into cognitive overhead in interactive problem solving. The key premise is that any hidden object could become relevant at any time as a function of a user's idiosyncratic knowledge structures interacting with cues in the environment. Clutter can be thought of as the cost of selecting a direction in which to look when an object does become relevant. This cost in turn has two terms: the cost of imagining where the object might be and the increased chance of becoming distracted in the process. These costs are mitigated by simplifying the interface in terms of number of directions in which to look, perhaps explaining the enduring appeal of one-dimensional virtual teletypes of the kind used in the current study.

6.3.2. The potential of browsing

The model suggests that memory depends on attention, not intent. For example, event chunks are stored in memory as a by-product of attending to an item, with no need for any specific intent to revisit that item later. The implication is that people store vast amounts of temporal "pointers" into their environment that they would recall given the right cues. In the model, this quantity is reflected in the imbalance between productions acquired and productions fired. Of the 1320 productions stored in 10 min, only 37% (491/1320) actually fire, accounting for only 9% of total firings during the simulation.

The implication for interactive behavior is that activities like browsing are potentially much better investments than we might have thought. Unlocking this potential would involve analysing the semantic structure of the knowledge being browsed and then asking how the interface might help produce good cues later when the browsed information would be relevant.

The potential of browsing rests on the assumption that encoding is triggered automatically as a side effect of attention. This assumption is in principle testable against alternative models in which encoding is more selective, for example by the kind of priming paradigm used in implicit memory (e.g. Jacoby, Toth & Yonelinas, 1993). Soar has been criticized for the "overfecundity" of its learning mechanism (Anderson & Lebiere, 1998, p. 448), but this criticism misinterprets an important predictive constraint. The chunking mechanism reformulates cue-dependent forgetting (Tulving, 1974) to predict that memory stores much more information about our environment than the environment typically helps us retrieve.

8. Conclusions

This article introduced the construct of near-term memory as a means of analysing memory in natural settings without sacrificing precision or detail. Near-term memory is analysed by developing an extended fine-grained process model of behavioral data, grounding the model in well-founded theoretical assumptions. The model is then interrogated for quantitative and structural insights into what was stored in and retrieved from memory.

The model described here incorporates assumptions of encoding specificity and automaticity on the cognitive side, and assumptions about volume of information and unpredictability of need on the task side. Taken together, these assumptions imply moment-by-moment storage of large amounts of detail of diverse kinds, including but not limited to perceptual chunks, recoded semantic knowledge, and event chunks pointing to objects in the environment. The model precisely defines each category in terms of what new symbolic associations are acquired by the system. Moreover, the model's representational gaps give an indication of what other learning processes are likely to operate, for example in the construction of situation models.

The general implication for expert problem solving is that it depends as much on episodic detail built up in memory during task performance as it does on stable semantic knowledge brought to the task. For HCI, a more specific implication is that interfaces can be evaluated on how they help or hinder access to relevant detail in memory, on the assumption that such access is mediated by semantic cues originally specific to the encoding context.

References

ALTMANN, E. M. (1996). *Episodic memory for external information*. Doctoral Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

ALTMANN, E. M. & GRAY, W. D. (1999). Serial attention as strategic memory, *Proceedings of the 21st Annual Meeting of the Cognitive Science Society*, pp. 25–30. Hillsdale, NJ: Erlbaum.

ALTMANN, E. M. & JOHN, B. E. (1999). Episodic indexing: a model of memory for attention events. *Cognitive Science*, **23,** 117–146.

ALTMANN, E. M., LARKIN, J. H. & JOHN, B. E. (1995). Display navigation by an expert programmer: a preliminary model of memory. In I. R. KATZ, R. MACK, L. MARKS, M. B. ROSSON & J. NIELSEN, Eds. *ACM CHI'95 Conference on Human Factors in Computing Systems*, pp. 3–10. New York: ACM Press.

ALTMANN, E. M. & TRAFTON, J. G. (1999). Memory for goals: an architectural perspective. *Proceedings of 21st Annual Meeting of the Cognitive Science Society*, pp. 19–24. Hillsdale: Erlbaum.

ANDERSON, J. R. & BOWER, G. H. (1972). Recogntion and retrieval processes in free recall. *Psycological Review*, **79,** 97–123.

ANDERSON, J. R. & BOWER, G. H. (1973). *Human Associative Memory*. Washington, DC: Winston.

ANDERSON, J. R. & LEBIERE, C. (Eds.). (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Erlbaum.

BALLARD, D. H., HAYHOE, M. M., POOK, P. K. & RAO, R. P. N. (1997). Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, **20,** 723–767.

BROOKS, R. E. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man–Machine Studies*, **9,** 737–751.

BROOKS, R. E. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man–Machine Studies*, **18,** 543–554.

CHASE, W. G. & SIMON, H. A. (1973). The mind's eye in chess. In W. G. CHASE, Ed., *Visual Information Processing*. New York: Academic Press.

DAVIES, S. P. (1994). Knowledge restructuring and the acquisition of programming expertise. *International Journal of Human-Computer Studies*, **40,** 703–726.

DELANEY, P. F., REDER, L. M., STASZEWSKI, J. J. & RITTER, F. E. (1998). The strategy–specific nature of improvement: the power law applies by strategy within task. *Psychological Science*, **9,** 1–7.

DOORENBOS, R. (1995). *Production learning for large learning systems*. Doctoral Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

EBBINGHAUS, H. (1885/1964). *Memory: a Contribution to Experimental Psychology*. Dover: New York.

ERICSSON, K. A. & DELANEY, P. F. (1999). Working memory in everyday skilled performance. In A. MIYAKE & P. SHAH, Eds. *Models of Working Memory: Mechanisms of Active Maintenance and Executive Control*, pp. 257–297. New York: Cambridge University Press.

ERICSSON, K. A. & KINTSCH, W. (1995). Long-term working memory. *Psychological Review*, **102,** 211–245.

ERICSSON, K. A. & LEHMANN, A. C. (1996). Expert and exceptional performance: Evidence of maximal adaptation to task constraints. *Annual Review of Psychology*, **47,** 273–305.

ERICSSON, K. A. & SIMON, H. A. (1993). *Protocol Analysis: Verbal Reports as Data*. (Revised edn). Cambridge, MA: The MIT Press.

GOLLEDGE, R. G. (1991). Cognition of physical and built environments. In T. GARLING & G. W. EVANS, Eds. *Environment, Cognition, and Action*. New York: Oxford University Press.

GRAY, W. D. (2000). The nature and processing of errors in interactive behavior. *Cognitive Science*, **24,** 205–248.

GRAY, W. D. & BOEHM-DAVIS, D. A. (in press). Milliseconds matter: an introduction to micro-strategies and to their use in describing and predicting interactive behavior. *Journal of Experimental Psychology: Applied*.

GRAY, W. D., JOHN, B. E. & ATWOOD, M. E. (1993). Project Ernestine: validating a GOMS analysis for predicting and explaining real-world performance. *Human-Computer Interaction*, **8,** 237–309.

GREEN, T. R. G., BELLAMY, R. K. E. & PARKER, J. M. (1987). Parsing and Gnisrap: a model of device use. In G. M. OLSON, S. SHEPPARD & E. SOLOWAY, Eds. *Empirical Studies of Programmers: Second Workshop*, pp. 132–146. Norwood, NJ: Ablex.

HOWES, A. & YOUNG, R. M. (1997). The role of cognitive architecture in modelling the user: Soar's learning mechanism. *Human–Computer Interaction*, **12,** 311–343.

HUTCHINS, E. (1995a). *Cognition in the Wild*. Cambridge, MA: MIT Press.

HUTCHINS, E. (1995b). How a cockpit remembers its speed. *Cognitive Science*, **19,** 265–288.

JACOBY, L. L., TOTH, J. P. & YONELINAS, A. P. (1993). Separating conscious and unconscious influences of memory: measuring recollection. *Journal of Experimental Psychology: General*, **122,** 139–154.

JEFFRIES, R., TURNER, A. A., POLSON, P. G. & ATWOOD, M. E. (1981). The processes involved in designing software. In J. R. ANDERSON, Ed. *Cognitive Skills and their Acquisition*. Hillsdale, NJ: Erlbaum.

JOHN, B. E. & ALTMANN, E. M. (1999). *The power and constraint provided by an integrative cognitive architecture. Paper Presented at the 2nd International Conference on Cognitive Science*. Tokyo, Japan, 27–30 July.

JOHN, B. E. & KIERAS, D. E. (1996). The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on Computer–Human Interaction*, **3,** 320–351.

KANT, E. & NEWELL, A. (1984). Problem solving techniques for the design of algorithms. *Information Processing and Management*, **20,** 97–18.

KINTSCH, W. (1970). Models for free recall and recognition. In D. A. NORMAN, Ed. *Models of Human Memory*. New York: Academic Press.

KINTSCH, W. (1998). *Comprehension: A Paradigm for Cognition*. New York: Cambridge University Press.

KIRWAN, B. & AINSWORTH, L. K., Eds. (1992). *A Guide to Task Analysis*. Washington, DC: Taylor & Francis.

LAIRD, J. E., ROSENBLOOM, P. S. & NEWELL, A. (1986). Chunking the Soar: the anatomy of a general learning mechanism. *Machine Learning*, **1,** 11–46.

LARKIN, J. H. (1989). Display-based problem solving. In D. KLAHR & K. KOTOVSKY, Eds. *Complex Information Processing: The Impact of Herbert A. Simon*, pp. 319–341. Hillsdale, NJ: Erlbaum.

LEHMAN, J. F., LAIRD, J. E. & ROSENBLOOM, P. S. (1998). A gentle introduction to Soar: An architecture for human cognition. In D. N. OSHERSON, Ed., *An Invitation to Cognitive science: Methods, Models, and Conceptual Issues*, Vol. 4. Cambridge, MA: MIT Press.

LEWIS, R. L. (1993). *An Architecturally-based Theory of Human Sentence Comprehension*. Carnegie Mellon University, Pittsburgh, PA.

LOHSE, G. L. & JOHNSON, E. J. (1996). A comparison of two process tracing methods for choice tasks. *Organizational Behavior and Human Decision Processes*, **68,** 28–43.

NEISSER, U. (1976). *Cognition and Reality: Principles and Implications of Cognitive Psychology*. San Francisco: W. H. Freeman and Co.

NEWELL, A. (1973). You can't play 20 questions with nature and win: projective comments on the papers of this symposium. In W. G. CHASE, Ed. *Visual Information Processing*, pp. 283–308. New York: Academic Press.

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

NEWELL, A. & SIMON, H. A. (1972). *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall, Inc.

OLSON, J. S. & NILSEN, E. (1988). Analysis of the cognition involved in spreadsheet software interaction. *Human–Computer Interaction*, **3,** 309–349.

PASHLER, H. E. (1997). *The Psychology of Attention*. Cambridge, MA: The MIT Press.

PENNINGTON, N. (1987). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, **19,** 295–341.

RIEMAN, J., LEWIS, C., YOUNG, R. M. & POLSON, P. G. (1994). "Why is a Raven Like a Writing Desk?" Lessons in interface consistency and analogical reasoning from two cognitive architectures. In B. ADELSON, S. DUMAIS & J. OLSON, Eds. *ACM CHI'94 Conference on Human Factors in Computing Systems*, Vol. 1, pp. 438–444. New York: ACM Press.

RIEMAN, J., YOUNG, R. M. & HOWES, A. (1996). A dual-space model of iteratively deepening exploratory learning. *International Journal of Human–Computer Studies*, **44,** 743–775.

RITTER, F. E. & BIBBY, P. A. (1997). *Modelling Learning as it Happens in a Diagrammatic Reasoning Task*, 45. Nottingham, UK: ESRC Centre for Research in Development, Instruction, and Training.

SCHNEIDER, W. & SHIFFRIN, R. M. (1977). Controlled and automatic human information processing: I. Detection, search, and attention. *Psychological Review*, **84,** 1–66.

SIEGLER, R. S. (1991). The microgenetic method: a direct means for studying cognitive development. *American Psychologist*, **46,** 606–620.

SUCHMAN, L. A. (1987). *Plans and Situated Action: The Problem of Human–Machine Communication*. New York: Cambridge University Press.

TULVING, E. (1974). Cue-dependent forgetting. *American Scientist*, **62,** 74–82.

TULVING, E. (1983). *Elements of Episodic Memory*. New York: Oxford University Press.

VENTURINO, M. (1997). Interference and information organization in keeping track of continually changing information. *Human Factors*, **39,** 532–539.

VON MAYRHAUSER, A., & VANS, A. M. (1996). Identification of dynamic comprehension processes during large scale maintenance. *IEEE Transactions on Software Engineering*, **22,** 424–437.

WATKINS, M. J. & GARDINER, J. M. (1979). An appreciation of generate-recognize theory of recall. *Journal of Verbal Learning and Verbal Behavior*, **18,** 687–704.

WIEDENBECK, S. (1991). The initial stage of comprehension. *International Journal of Man-Machine Studies*, **35,** 517–540.

YNTEMA, D. B. (1963). Keeping track of several things at once. *Human Factors*, **5,** 7–17.

YOUNG, R. M. & LEWIS, R. L. (1999). The Soar cognitive architecture and human working memory. In A. MIYAKE & P. SHAH, Eds. *Models of Working Memory*: *Mechanisms of Active Maintenance and Executive Control*, pp. 224–256. New York: Cambridge University Press.

ZHANG, J. & NORMAN, D. A. (1994). Representations in distributed cognitive tasks. *Cognitive Science*, **18,** 87–122.

ZSAMBOK, C. E. & KLEIN, G., Eds. (1997). *Naturalistic Decision Making*. Mahwah, NJ: Erlbaum.