Experiences with Novices: The Importance of Graphical Representations in Supporting Mental Models

Carlisle E. George

School of Computing Science
Middlesex University, London
c.george@mdx.ac.uk

Keywords: POP-II.A. Novice/Expert, POP-III.A. Recursion, POP-V.A. Mental Models

Abstract

Recursion is an important problem solving technique used in programming. It is also a highly unfamiliar mental activity and many computing novices have difficult understanding recursion and applying recursive techniques in problem solving. Research studies have concluded that novices and experts differ in their mental models of recursion. Novices seem to possess various inadequate models of recursion especially the iterative or *loop* model. This paper examines whether novices who are aided in acquiring an expert's mental model of recursion (the *copies* model) can effectively use this model in evaluating recursive algorithms. Results of a study indicated that a large percentage of novices who had previously demonstrated an understanding of the *copies* model (using explicit diagrammatic traces) failed do so when not using diagrammatic traces. In fact, they appeared to demonstrate evidence for the incorrect iterative or *loop* model when trying to mentally evaluate recursive programs. The results provide evidence that mental models are unstable and that graphical representations are a very necessary aid to retrieval of novices' mental models. This suggests that the teaching of recursion may be best facilitated by teaching students how to simulate the execution of a recursive algorithm using diagrammatic traces.

Introduction

Cognitive research studies in computer programming and other disciplines have demonstrated the importance of adequate mental representations or models in understanding processes, complex tasks, or systems and have also revealed that learning is facilitated by aiding learners in forming appropriate mental models or by modifying and elaborating learners' incomplete models (e.g. DuBoulay, O'Shea & Monk 1980; Mayer 1981). While novices may acquire an understanding of the syntax of a programming language, without appropriate mental models they are unable to successfully understand complex concepts or engage in more difficult aspects of problem solving. Effective mental models are thus of critical importance in understanding programming concepts.

This paper focuses on recursion, a programming concept which many students find difficult to understand and to apply in their problem solving activities (e.g. Anderson et al 1988; Anzai & Uesato, 1982a, 1982b; Baird 1986; Elenbogen & O'kennon 1988; Ford, 1984; Ginat & Shifroni 1999; Henderson & Romero 1989; Kahney 1983; Kessler & Anderson 1986; Kurland & Pea 1983; Pirolli, 1986; Roberts 1986; Wiedenbeck, 1988). Recursion is a misunderstood or barely understood programming concept among programmers (Ford 1982). It is difficult because it is a highly unfamiliar mental activity (Anderson et al 1988) and may be counter-intuitive to beginning computer science students (Elenbogen & O'Kennon 1988).

Kahney (1983) concluded that novices and experts differ in their mental models of recursion and further categorised various mental models that novices were seen to possess. Kahney's conclusions have been supported by many other subsequent studies (cited above). They indicate that most novices, unlike experts, are unable to conceptualise separate and unique invocations of subprograms in recursion (the *copies* model) and more importantly the flow of control in an executing recursive program. Novices easily adopt the more familiar notion of iteration when attempting to understand recursion (Kessler & Anderson 1986; Kurland & Pea 1983; Wiedenbeck, 1989).

In Kahney's study, students were not explicitly taught how to simulate the flow of execution control and visualise separate invocations of subprograms. They were only required to comment on whether the solution programs given would work and did not seemed to have used any diagrammatic drawings to help them in their problem-solving. This paper will investigate whether novices who are aided in acquiring an expert's mental model of recursion (the *copies* model) can effectively use this model in evaluating recursive algorithms. Particular focus will be placed on the role of graphical /diagrammatic execution traces during problem solving.

Mental Models

The history of the concept of mental models in human cognition may be traced to Craik (1943), who suggested that humans make use of internal models of external reality, which enable them to better understand and react to situations in their environment. These internal models are formed through interaction with external

events. He argues that humans reason by manipulating internal symbolic representations and translating them back into actions or at least recognising the correspondence between these internal representations and external events. Eysenck & Keane (1990) make the distinction between two bodies of research into mental models: (i) mental models viewed as analogical representations distinguished from propositional representations and images - e.g work by Johnson-Laird (1983) which is primarily concerned with analogy and insight problems involving classical syllogisms *and* (ii) mental models as theoretical constructs, characterised in propositional terms, and used to account for various aspects of behaviour especially in novel problem solving situations. - e.g. Gentner & Stevens, (1983) and Norman (1983) whose work has focused on investigating everyday problem situations where people used their 'mental models' to understand the world, especially physical systems (also see Bennett 1984; Kieras & Boviar 1984; Carroll & Olson 1987; Sein 1988; Gentner & Gentner 1983).

This paper is primarily concerned with the 'mental models' as theoretical constructs and for purposes of this work, 'mental models' generally refer to a learner's mental representations or knowledge of processes, complex tasks or systems (formed from interactions with abstract or concrete representations) which on construction and manipulation allows the learner to reason, predict and understand the particular process, task or system.

Kahney's Mental Models of Recursion

Kahney (1983) attempted to test the hypothesis that novice programmers and experts differ in terms of their respective mental models of recursion as a process and further tried to discriminate between the models of recursion which novices actually possessed. Experts were hypothesised to possess a *copies* model of recursion while novices were hypothesised to possess a *loop* model. In the *copies* model, a recursive subroutine or subprogram is defined as a process which is capable of triggering new instantiations (or copies) of itself with control passing forward to successive instantiations and back to suspended ones. In the *loop* model a recursive subroutine is viewed as a single object (instead of a series of new instantiations) consisting of an entry point, an action part and a propagation mechanism.

The hypothesis about the differences between novice and expert models was tested by presenting subjects with a questionnaire containing three programs written in the SOLO programming language (Eisenstadt 1978). Subjects were asked to say whether the programs would perform a specific task and if not, to explain why. Using a database containing the names of persons who kiss each other, the programs were required to make the inference that: if somebody 'X' has flu, then whoever 'X' kisses also has flu, and whoever is infected spreads the infection to the person he/she kisses, and so on. The resulting output of the correct program(s) was a new database showing all persons infected with flu.

The three programs differed in their actions. Program-1 terminated after infecting the first name passed as an argument from the database and hence did not achieve the required effect. Program-2 and Program-3 both performed the required task but differed in procedure. Program-2 (an example of tail recursion) worked by 'infecting' the first argument passed then generating the next argument hence triggering recursion. Program-3 (an example of embedded recursion) worked by first creating a stack of bindings for the arguments and 'infecting' each on return from the recursive creation of the list, hence infected each argument in reverse order to Program-1. Figure 1 below shows the three programs written in the SOLO programming language. The asterisk (*) represents an argument (a name) from the database.

Program-1

```
 TO  INFECT /X/
 1  CHECK /X/ KISSES?
  1A  If  Present: NOTE *  HAS FLU; EXIT
  1B  If  Absent: EXIT
DONE
```

Program-2

```
 TO  INFECT /X/
 1  NOTE /X/ HAS FLU
2  CHECK /X/  KISSES?
  2A  If  Present: INFECT * ; EXIT
  2B  If  Absent: EXIT
DONE
```

Program-3

```
TO  INFECT /X/
 1  CHECK /X/ KISSES?
   1A  If  Present: INFECT *  ; CONTINUE
   1B  If  Absent: CONTINUE
 2 NOTE /X/ HAS FLU
 DONE
```

Figure 1 - Text of Kahney's SOLO programs

Strong evidence for possession of the *copies* model of recursion was taken to be the selection of Program-1 and Program-2. Strong evidence for possession of the *loop* model of recursion was taken to be selection of Program-2 and rejection of Program-3 on the grounds that only the last argument would be infected. Various other criteria relating to written explanations determined the extent of possessing idiosyncratic notions of recursion and also provided insight into the possible existence of other competing models. Questionnaires from 30 subjects classified as 'respondents' and 9 expert programmers were analysed.

Results showed a highly significant result in the difference in selection between novices and experts (chi-squared = 21.40, $p < .001$). Novices chose Program-1 and Program-3 significantly less often than experts (chi-squared = 10.78, $p < .01$). From the data, eight of the nine experts showed strong evidence for the *copies* model of recursion and only one novice (3%) showed evidence for a *copies* model of recursion. Kahney found evidence

for the *loop* model of recursion among novices (53%) but also concluded after further analysis that for processes like recursion, different novices may acquire a wide range of mental models, may achieve no understanding at all or may simply be able to identify a program as a member of a particular class if it has expected features in expected configurations. He identified three other possible models of recursion which were different from either the *copies* model or *loop* model.

Respondents who said that none of the programs would work were referred to as having a *null model* or no model of recursion. Kahney found evidence which suggested that some respondents had slightly idiosyncratic *copies* or *loop* models and hence did not correctly predict the behaviour of the programs. He referred to this group as having an *odd model* of recursion. Among other observations, a common notion acquired in this group was that the flow of control statement (e.g. EXIT, Figure 3.1) rather than the absence of a particular pattern in the database acts as the stopping rule for recursion. Finally Kahney identified the *syntactic* or *magic model*, where respondents recognised a particular program structure as being a recursive procedure based on previously seen recursive programs. Although they knew what the procedure did, they had no idea of the actual behaviour of the process, or how it achieved its effects. Recursion thus remained a mystery.

Revisiting Kahney

George(1996) developed the EROSI tutor which was used to aid novices in acquiring the *copies* model of recursion by first helping novices understand a sophisticated subprogram execution model called the *dynamic logical* model of subprogram calls which is then linearly extended to self referential calls resulting in the *copies* model of recursion. As part of a Final Study, after using the EROSI tutor and learning how to diagrammatically trace the copies execution model of recursion, forty nine novices were given post-treatment tasks which tested their ability to evaluate and construct recursive programs. All students were advised to use diagrammatic traces during their problem solving activities. A few days after completing the post-treatment tasks, twenty-two students were interviewed.

One part of the post-treatment interviews focused on investigating subjects' mental models of recursion by using a modified version of Kahney's Model Test (see Appendix A). Twenty-one of the twenty-two students who were interviewed, volunteered to do Kahney's model test. On arrival at the interview, each student was given one of four textual variants of the test (to avoid collusion) to study for about five to ten minutes before he/she was asked to give answers to the questions. Consistent with the criteria used by Kanhey, from three solutions given (entailing: no recursion, tail recursion, embedded recursion), evidence for the possession of the *copies* model was identification of the last two solutions as being correct and the ability to identify differences in their output sequences. Evidence for a *loop* or iterative model was identification of the second solution only as being correct and rejection of the third solution on the basis that only the last person infected would get flu. Evidence of a 'null' model or no understanding of recursion was the inability to identify either of the last two solutions as being correct or identifying the first solution as being correct.

Analysis of Performance on Kahney's Model Test.

Some students had difficulty understanding either the problem to be solved or some other aspect of the test, however, after some explanations they were able to give their answers. All students were able to identify that the first solution did not involve recursion and was therefore incorrect. Eight students (38%) showed evidence for possessing the *copies* model, and thirteen students (53%) showed evidence for the *loop* model. All of the eight students who showed evidence for the *copies* model had previously demonstrated a good understanding of the *copies* execution of an embedded recursive algorithm. Conversely only one of the students who showed evidence for a *loop* model had made an analogous error in evaluating the embedded recursive algorithm. Moreover nine (out of thirteen i.e. 69%) of them had previously shown a good understanding of the *copies* execution of the embedded recursive algorithm.

These results point to the difference in performance when using diagrammatic traces as opposed to mentally simulating the execution of recursion. Some students thought that they would be more certain of their answer if they had taken the time to draw diagrams. For example when one student was pressed to give a definite statement after saying that he was not completely sure of his answer he said: '*I don't feel confident just to say, it just needs a couple of drawings really*'. Also when shown what the correct answer was, a student who had previously shown evidence for a loop model remarked: '*Oh! Yes I can see that. I think that if I had drawn boxes I would have been able to work it out, but it's so hard to think about it without drawing the boxes'*. These observations seem to highlight the importance of teaching students how to explicitly simulate recursive processes using diagrammatic traces.

Discussion

Results of Kahney's Model Test compared to the Post-treatment tasks showed that in the absence of explicit diagrammatic traces many students who evaluated embedded recursive algorithms mentally, gave answers which were indicative of a *loop* model of recursion. With diagrammatic traces, however, these students did not make the same error. In fact on the post-treatment tasks only two students , who incidentally did not use any diagrams when evaluating an embedded recursive algorithm, gave incorrect responses corresponding to the use of a *loop* model of recursion.

A possible explanation for the results above may be found in the work of Norman(1983), who observed that mental models may be 'unstable' (forgotten or confused with similar systems) and that people usually keep them as uncomplicated as possible, avoiding mental complexity. With respect to the *copies* model, the above seems to apply when diagrammatic traces are not used. This implies that perhaps the teaching of recursion is best accompanied by the teaching of explicit evaluation methods involving diagrammatic traces of recursive

processes. Various explanations for the effectiveness of graphical representations in reasoning and problem-solving include that they: facilitate perceptual judgements and act as aids to retrieval (Larkin & Simon, 1987); *and* reduce search and working memory load by organising information by location (Cox & Brna, 1995).

In reviewing the work of some previous studies (e.g. Kahney, 1983) which reported that novices generally possess a loop (or iterative) model for recursion, the findings above can perhaps illuminate a major reason for these observations. Generally from reports of these studies students were either not taught how to diagrammatically simulate the execution of a recursive algorithm or were not encouraged to do so during their problem-solving activities. Therefore they probably mentally simulated the process of recursion, in the process encountering difficulty with keeping track of variables and flow of control, and hence adopted the easy and more familiar iterative model.

References

Anderson, J. R., Pirolli, P., & Farrell, R. (1988). Learning to program recursive functions. In M. T. Chi, R. Glaser, & M. J. Farr. (Eds.), *The Nature of Expertise*, 151-183. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anzai, Y., & Uesato, Y. (1982a). Is recursive computation difficult to learn? *CIP paper No.439, Dept. of Psychology, Carnegie-Mellon University*, 1982.

Anzai, Y., & Uesato, Y. (1982b). Learning recursive procedures by middleschool children. *Proceedings of the Fourth Annual Conference of the Cognitive Science Society, Ann Arbor, Michigan*, 100-102, August 1982.

Baird, W.G. (1986). My freshmen learn recursion, *ACM SIGCSE BULLETIN*, 18(2), June 1986, 25-28.

Bennett, K. B. (1984). The Effect of Display Design on the User's Mental Model of a Perceptual Database System. (Doctoral dissertation, The Catholic University of America, 1984). *Dissertation Abstracts International, 45*, 1604B.

Bhuiyan, S. H. (1992). *Identifying and Supporting Mental Methods of Recursion in a Learning Environment.* Unpublished PhD Thesis. Dept. of Computational Science, University of Saskatchewan. Saskastoon, Canada.

Carroll, J. M., & Olson, J. R. (Eds.), (1987). *Mental Models in Human-Computer Interaction: Research Issues about what the User of Software Knows*. Washington, DC: National Academy Press.

Cox, R., & Brna, P. (1995). Supporting the use of external representations in problem solving: The need for flexible learning environments. *Journal of Artificial Intelligence in Education* 1995, 6(2/3), 239-302.

Craik, K. (1943). *The Nature of Explanation*. UK: Cambridge University Press.

Du Boulay, B., O'Shea, T., & Monk, J. (1981). The black box inside the glass box. *International Journal of Man-Machine Studies*, 14, 237-249.

Elenbogen, B. S., & O'Kennon, M. R. (1988). Teaching recursion using fractals in PROLOG. *ACM SIGCSE BULLETIN 20(1)*, Feb.,1988, 263-266

Eisenstadt, M. (1978). The SOLO primer. Units 3-4, *Cognitive Psychology: A Third Level Course*. Milon Keynes UK: Open University Press.

Eysenck, M. W., & Keane, M. T. (1990). *Cognitive Psychology: A Student's Handbook.* London, UK: Lawrence Erlbaum Associates.

Ford, G. (1982). A framework for teaching recursion. *ACM SIGCSE BULLETIN* 14(2), June 1982, 32-39.

Ford, G. (1984). An implementation-independent approach to teaching recursion, *ACM SIGCSE BULLETIN*, 16(1), Feb.,1984, 213-216.

Gentner, D. (1983). Structure-mapping: a theoretical framework for analogy. *Cognitive Science*, 7, 155-170.

Gentner, D., & Gentner, D. R. (1983). Flowing waters or teemings crowds: mental models of electricity. In D. Gentner & A. L. Stevens (Eds), *Mental Models*, 99-129. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Gentner, D., & Stevens A. L. (Eds), (1983). *Mental Models*, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

George, C.E. (1996). *Investigating the Effectiveness of a Software-Reinforced Approach to Understanding Recursion*. PhD Thesis, University of London, Goldsmiths.

Ginat & Shifroni (1999). Teaching Recursion in a Procedural Environment – How much should we emphasize the computing model? *Proceedings of ACM SIGCSE '99*, p 127 – 131.

Henderson, P. B., & Romero, F. J. (1989). Teaching recursion as a problem-solving tool. *ACM SIGCSE BULLETIN*, Feb., 1989, 21(1), 27-31.

Johnson-Laird, P. N. (1983). *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness*. Cambridge, UK: Cambridge University Press

Kahney, H. (1983). What do novice programmers know about recursion?. *Proceedings of the CHI '83 Conference on Human Factors in Computer Systems*, 235-239. Boston, MA.

Katz, N. (1986). Construct validity of Kolb's Learning Style Inventory, using factor analysis and Guttman's smallest space analysis. *Perceptual and Motor Skills*, 63, 1323-1326.

Kessler, C. M., & Anderson, J. R. (1986). Learning flow of control: Recursive and iterative procedures. *Human-Computer Interaction*, 2, 135-166.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science*, 8, 255-273.

Kurland, D. M., & Pea, R. D. (1983). Children's mental models of recursive LOGO programs. *Proceedings of the 5th Annual Conference of the Cognitive Science Society*, session 4, 1-5. Rochester, NY.

Larkin, J.H., & Simon, H.A. (1987). Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, 11, 65-100.

Mayer, R. (1981). The psychology of how novices learn computer programming. *Computing Surveys*, 13(1), 121-141.

Norman, D. A. (1983). Some observations on mental models. In D. Gentner & A. L. Stevens (Eds.), *Mental Models*, Hillsdale, NJ: Lawrence Erlbaum Associates Inc.

Pirolli, P. L. (1985). *Problem Solving by Analogy and Skill Acquisition in the Domain of Programming*. Unpublished doctoral dissertation, Carnegie-Mellon University, Pittsburgh.

Pirolli, P. (1986). A Cognitive model and computer tutor for programming recursion. *HUMAN-COMPUTER INTERACTION*, 2, 319-355.
Roberts, E. S. (1986).*Thinking Recursively*. NY: John Wiley & Sons, Inc.
Sein, M. K. (1988). Conceptual Models in Training Novice Users of Computer Systems: Effectiveness of Abstract vs. Analogical Models and Influence of Individual Differences. (Doctoral dissertation, Indiana University). *Dissertation Abstracts International*, 49, 880A.
Wiedenbeck, S. (1988). Learning recursion as a concept and as a programming technique. *ACM SIGCSE BULLETIN 20(1)*, Feb., 1988, 275-278.
Wiedenbeck, S. (1989). Learning iteration and recursion from examples. *International Journal of Man-Machine Studies*, 30, 1-22.

Appendix A

Modified Kahneys Model test questionnaire (Interview)

A doctor in a small town knows the following secret:

Jack $\overset{\textbf{kisses}}{\rightarrow}$ **Fay** $\overset{\textbf{kisses}}{\rightarrow}$ **Tom** $\overset{\textbf{kisses}}{\rightarrow}$ **Sue** $\overset{\textbf{kisses}}{\rightarrow}$ **Chris**.

Contagious Rule:

If a person X is infected with flu *and* kisses person Y then person Y will become infected with the flu and *whoever* person Y kisses becomes infected with the flu and so on.

I want to write a procedure* INFECT *to infect a person and to obey the contagious Rule.

My procedure must return the names of every person who has the flu. Hence **INFECT(Jack)** should tell me that**:**

Jack *has flu*; Fay *has flu*; Tom *has flu*; Sue *has flu*; Chris *has flu*. (***In any order***).

Assume that I have the function **Kisses**(X) which returns the name of the person whom X kisses. If X does not kiss anyone then the function returns the string 'Nobody'. Hence

Kisses(Jack) returns 'Fay'
Kisses(Fay) returns 'Tom'.
Kisses(Tom) returns ' Sue'
Kisses(Sue) returns 'Chris'
Kisses(Chris) returns 'Nobody'

I have been given three solutions for procedure **INFECT**. Consider each of the solutions given below in turn and say:

(a) Whether or not it will give me the names of all persons infected with the flu. *and*

(b) If it will **say how** it does it (*in your own words*) or if it won't, **say why** it doesn't (*again in your own words*).

Solution Blue

```
PROCEDURE   INFECT( X : String );
 VAR   Y : String;
 BEGIN
   Y := Kisses(X);
    IF  Y<> 'Nobody' THEN  Write( Y, ' has Flu' );
 END;
```

Solution Black

```
PROCEDURE    INFECT( X : String );
 VAR   Y : String;
 BEGIN
  Write(X, ' has Flu');
   Y := Kisses(X);
   IF  Y<> 'Nobody' THEN  INFECT (Y)
 END;
```

Solution Green

```
PROCEDURE    INFECT( X : String );
 VAR   Y : String;
 BEGIN
   Y := Kisses(X);
    IF  Y<> 'Nobody' THEN  INFECT(Y);
    Write(X,  'has Flu')
 END;
```
