ARI Research Note 95-40

Transfer of Skills Among Programming Languages

John R. Anderson
Carnegie Mellon University

Research and Advanced Concepts Office
Michael Drillings, Acting Chief

June 1995

United States Army
Research Institute for the Behavioral and Social Sciences

U.S. ARMY RESEARCH INSTITUTE FOR THE BEHAVIORAL AND SOCIAL SCIENCES

A Field Operating Agency Under the Jurisdiction of the Deputy Chief of Staff for Personnel

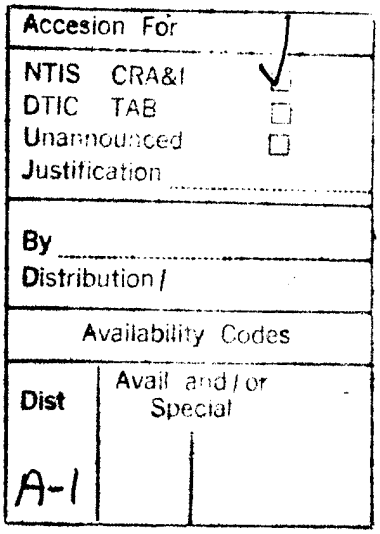
Edgar M. Johnson
Director

Research accomplished under contract
for the Department of the Army

Carnegie Mellon University

Technical review by

Joseph Psotka



NOTICES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	1995, June	FINAL 8/89 - 8/94

4. TITLE AND SUBTITLE

Transfer of Skills Among Programming Languages

5. FUNDING NUMBERS

MDA 903-89-K-0190
0601102A
B74F
C04
2901

6. AUTHOR(S)

John R. Anderson (Carnegie Mellon University)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Psychology
Baker Hall 346-B
Carnegie Mellon University
Pittsburgh, PA 15213

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Institute for the Behavioral and Social Sciences
ATTN: PERI-BR
5001 Eisenhower Avenue
Alexandria, VA 22333-5600

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARI Research Note 95-40

11. SUPPLEMENTARY NOTES

COR: Michael Drillings

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The general picture that has emerged from this research is one in which programming skill is to be conceived as translation from one surface representation to another. While the successful student will have this surface representation annotated with rich representation of its functionality, the skill is still quite specific to the notational details of the representations involved. The initial context for this research was set by two things supported by a prior ARI contract. One of these was the development of a general theory of transfer of cognitive skill, which could be conceived as a modern information-processing rendition of Thorndike's theory of identical elements (Thorndike & Woodworth, 1901; Singley & Anderson, 1989). We showed that the degree of transfer could be predicted by the amount of overlap between knowledge structures in the ACT theory, which proposed that knowledge both consisted of procedural knowledge and declarative knowledge (Anderson, 1993). The other part of the research background for this project was the development of tutors for programming languages, particularly LISP (Anderson, Conrad, & Corbett, 1989). We wanted to generalize our understanding of both tutoring and of programming.

14. SUBJECT TERMS

tutoring procedural knowledge declarative knowledge skill transfer

15. NUMBER OF PAGES

14

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

TRANSFER OF SKILLS AMONG PROGRAMMING LANGUAGES

CONTENTS

Final Report for MDA903-89-K-0190
Transfer of Skills among Programming Languages
August 7, 1989 - August 7, 1994

John R. Anderson
Carnegie Mellon University

Background

The initial context for this research was set by two things supported by a prior ARI contract. One of these was the development of a general theory of transfer of cognitive skill (Singley & Anderson, 1989). This theory could be conceived as a modern information-processing rendition of Thorndike's theory of identical elements (Thorndike & Woodworth, 1901). Thorndike had proposed that one skill transferred to another skill as a function of the degree that they shared elements in common. Thorndike was never very precise on the nature of the shared elements but encouraged a general conception of them in terms of S-R bonds. This led to an overly narrow conception of what transfer would be obtained. Singley and Anderson showed that transfer could be much broader than this conception implied. However, we showed that the degree of transfer could be predicted by the amount of overlap between knowledge structures in the ACT theory (Anderson, 1993). The ACT theory proposed that knowledge both consisted of procedural knowledge and declarative knowledge. Procedural knowledge is embodied in production rules which leads to a "skill-like" competence that can apply when certain goals are set across a wide range of domains. Declarative knowledge is represented as facts or chunks (Anderson, 1993) which enable transfer of knowledge to serve other goals although the fluidity of transferred performance was not as great as when production rules serve as the basis of transfer. Basically, procedural knowledge has high performance and low transfer while declarative knowledge has low performance and high transfer.

The other part of the research background for this project was the development of tutors for programming languages, particularly LISP (Anderson, Conrad, & Corbett, 1989; Anderson & Reiser, 1985). We wanted to generalize our understanding of both tutoring and of programming. This led to the goal of developing a tutoring architecture that would support programming in multiple programming languages. This would both generalize our understanding of tutoring and provide us with a tool for studying transfer among multiple programming languages. Programming seemed an appropriately complex domain for understanding the procedural and declarative aspects of transfer postulated in the Anderson and Singley theory. We were not disappointed in this expectation.

Work on Tutoring Architectures

We initially developed a system for tutoring the programming languages LISP, Prolog, and Pascal but this architecture became generalized to support tutoring systems in general and became the basis of our tutoring work on high school mathematics. A special case of this tutoring architecture remains for developing programming tutors based on a structural editor interface. The work on the tutoring architecture is reported in Anderson and Pelletier (1991), Corbett, Anderson, and Fincham (1991), and Anderson, Corbett, Fincham, Hoffman, and Pelletier (1992). A more general overview of our work on tutoring is contained in Anderson, Corbett, Koedinger, and Pelletier (in press). All of these papers are included with this report.

We developed what we have come to call the Tutor Development Kit (TDK). Developing a tutor within it involves the following steps.

1. Create an interface for displaying the target competence. This can be done using LISP-like facilities provided by the TDK but one can also use independent software packages that communicate with the kit. It became

clear that the structure of the tutorial interface substantially determined the scope of the transfer.

2. Develop production rules which are capable of solving the target class of problems within the kit. Also create bug-rules to the represent student misconceptions. These production rules are relatively specific to the interface being used.

3. Attach declarative instruction to the target rules and bug rules and to the declarative representations of individual problems. These are represented as dialog templates which can be combined to create context-specific instruction.

4. Create a curriculum which amounts to creating a set of problems and establishing a set of mastery principles for controlling sequencing through the problems. We can deploy these tutors in a number of modalities achieving different divisions of control between tutor and student.

The tutor development kit provides the facilities for creating, interpreting, and debugging the production rules, integrating the various parts above, and for logging and analyzing data on student interactions. To date well over 2000 students have taken courses which involve many tens of hours of interaction with tutors created within this kit. This year over 500 Pittsburgh students spend half of their time in their algebra class interacting with our tutors. While ARI did not support creation of the algebra tutors, it did support initial creation of the underlying architecture.

Studies of Programming Transfer

While the majority of our research has involved study of transfer among programming languages, Wu and Anderson (1993, report enclosed) did study transfer among various strategies for iteration within the programming language

Pascal. There we found that subjects were very sensitive to the problem features that predicted the appropriateness of a particular programming strategy and did not suffer negative transfer from practicing other iteration constructs. This showed that students have high sensitivity to the functionality of programming constructs which is an important counterpoint to our other research showing their sensitivity to the surface form of the programs.

Our original approach to the main project was guided by the view that transfer among programming languages would be facilitated if we could get students to extract the "declarative essence" of a program. We hypothesized that this could be achieved by using a data-flow representation of the language. Therefore, we created a data-flow programming language and looked at students learning with it compared to their learning LISP. Some of this research is reported in Anjaneyulu and Anderson (1992) which is also enclosed with this report. In fact, we did not find any special advantages of this representation and found subjects treated it like any other programming language. In subsequent research we found that subjects basically did not have any representation of the essence of a programming language but rather had a number of superficial representations such as natural language, flowcharts, or actual code and that their skill was in translating between these representations. This was one piece of evidence among others that has led us to a view of competence much more tied to surface representation.

We followed this up with two directions of research concerned with transfer among specific languages. One direction of research looked at students who already knew how to program in two languages. They solved a problem in one language and then we looked at how well they transferred to solving the problem in a second language (Wu's 1992 dissertation, enclosed). This research found large positive transfer of the general algorithm but not of specific code. Thus, we found subjects were facilitated on the second program in terms of reduced planning time but not in terms of reduced coding time. Moreover,

subjects seemed to be carrying over the algorithm that best solved the problem in the first language. Thus, if they solved a problem in Prolog where a different algorithm was more appropriate they would tend to use that same algorithm in LISP. Also subjects showed a strong tendency to carry over superficial features like variable names from one program to another. This supported a view that subjects were transferring from a memory of the original code and setting themselves the task of writing equivalent code in the target language. Once again we see evidence for a notation-specific view of competence. They understand the functionality of the notation and certainly are not just doing symbol-for-symbol translations. Still that notation is the skeleton on which their functional understanding is attached.

More of our research has been focused on a different transfer paradigm. In this paradigm subjects are first taught one language and then another language. Early reports on this are Wu and Anderson (1991) and Anderson, Conrad, Corbett, Fincham, Hoffman, and Wu (1993), both of which are enclosed. There we found striking evidence for no transfer from learning the syntax of any one of LISP, Prolog, or Pascal and to learning the syntax of any other of these languages. Subjects showed no or little benefit when measured by coding time or coding errors. This turns out to actually be predicted by the knowledge representations in our tutors since the production rules used for any programming language in the tutors are completely different than the production rules for any other programming language. This reinforces the impression from other work that superficial differences in code are incorporated in the representation of the competence.

A much deeper analysis of the transfer of programming within our tutors was instigated by Leon Harvey and is reported in Harvey and Anderson (in press, also enclosed). This research looked at transfer between from the first lesson of Prolog to the first lesson of LISP. We replicated the earlier results of no transfer in terms of coding time but found large positive transfer in terms of time to read

the declarative instruction about LISP in the text that preceded the programming exercises in the tutor. We also found large positive transfer in time to read the instructional messages that the tutor delivered. Moreover, we were able to show that this transfer was specific to those portions of text that described concepts that were common between LISP and Prolog. A reading time model was developed that accurately predicted the degree of transfer to reading various passages. This indicated that there is a level of declarative knowledge involving things such as variables and list structures which does transfer between languages.

In the last year of the ARI contract, Al Corbett and I have followed up this research in two ways. First, we have established in a large course that this pattern of declarative but not procedural transfer holds up both going from LISP to Prolog and well as from Prolog to LISP. We have also established that there is declarative transfer for all lessons and not just the first lesson. Second, we have been using this analysis to facilitate instruction by creating special instructional modules to teach core declarative concepts. In particular, we have been focusing on the benefits of prior instruction on list structures. We show that subjects are then better able to learn a programming language and display more systematic learning curves. This research is still being written up. It remains to be seen whether improved declarative instruction results in any different patterns of transfer.

Conclusions

The general picture that has emerged from this research is one in which programming skill is to be conceived as translation from one surface representation to another. While the successful student will have this surface representation annotated with a rich representation of its functionality, the skill is still quite specific to the notational details of the representations involved. Thus, we do not see transfer of coding skills among programming languages. On the other hand, these representations have a common functionalities

involving things like variables, list structures, and iteration. An initial understanding of these functionalities and the natural language terms for describing them is something that can transfer among programming languages. We believe this conception of competence and transfer is not unique to programming but extends to other domains like mathematical problem solving.

This research can be interpreted fairly well in the Singley and Anderson framework set up in 1989. The coding skills are fundamentally procedural and because of representational differences involve production rules that cannot transfer across programming languages. On the other hand, the concepts and language of program functionality are general and will transfer across programming languages. The major evolution in our thinking since 1989 has concerned the importance of surface notation. While we are hardly at the level of Thorndike's S-R bonds, we clearly have moved in the direction of something closer to that original conception.

References

Anderson, J. R. (1993). Rules of the mind. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1993). The LISP tutor and skill acquisition. In J. R. Anderson (Ed.), Rules of the mind. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Conrad,, F. G., Corbett, A. T., Fincham, J. M., Hoffman, D., & Wu, Q. (1993). Computer programming and transfer. In J. R. Anderson (Ed.), Rules of the mind. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Corbett, A., Fincham, J., Hoffman, D., & Pelletier, R. (1992). General principles for an intelligent tutoring architecture. In V. Shute, & W. Regian (Eds.), Cognitive approaches to automated instruction (pp. 81-106). Hillsdale, NJ: Erlbaum.

Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (in press). Cognitive tutors: Lessons learned. The Journal of Learning Sciences.

Anderson, J. R., & Pelletier, R. (1991). A development system for model-tracing tutors. Proceedings of the International Conference of the Learning Sciences (pp. 1-8). Evanston, IL.

Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor. Byte, 10, 159-175.

Anjaneyulu, K. S. R., & Anderson, J. R. (1992). The advantages of data flow diagrams for beginning programming. Proceedings of the Second International Conference on Intelligent Tutoring Systems, Montreal.

Corbett, A. T., Anderson, J. R., & Fincham, J. M. (1991). Menu selection vs. typing: Effects on learning in an intelligent programming tutor. Proceeding of the International Conference of the Learning Sciences (pp. 107-112). Evanston, IL.

Harvey, L., & Anderson, J. R.. (in press). Transfer of declarative knowledge in complex information processing domains.

Singley, M. K., & Anderson, J.. R. (1989). Transfer of cognitive skill. Cambridge, MA: Harvard University Press.

Thorndike, E. L., & Woodworth, R. S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. Psychological Review, 9, 374-382.

Wu, Q. (1992). Knowledge transfer among programming languages. (Doctoral dissertation, Pittsburgh, PA: Carnegie Mellon University).

Wu, Q., & Anderson, J. R. (1991). Knowledge transfer among programming languages. Proceedings of the 13th Annual Conference of the Cognitive Science Society (pp. 376-381).

Wu, Q., & Anderson, J. R. (1993). Strategy choice and change in programming. International Journal of Man and Machine Studies, 39, 579-598.

Appendix

Publications Supported by Contract MDA903-89-K-0190 Transfer of Skills Among Programming Languages

Anderson, J. R., Conrad, F. G., Corbett, A. T., Fincham, J. M., Hoffman, D., & Wu, Q. (1993). Computer programming and transfer. In J. R. Anderson (Ed.), Rules of the mind. Hillsdale, NJ: Erlbaum.

Anderson, J. R., Corbett, A., Fincham, J., Hoffman, D., & Pelletier, R. (9192). General principles for an intelligent tutoring architecture. In V. Shute and W. Regian (Eds.), Cognitive approaches to automated instruction (pp. 81-106). Hillsdale, NJ: Erlbaum.

Anderson, J. R., Corbett, A. T., Koedinger, K., & Pelletier, R. (in press). Cognitive tutors: Lessons learned. The Journal of Learning Sciences.

Anderson, J. R., & Pelletier, R. (1991). A development system for model-tracing tutors. Proceedings of the International Conference of the Learning Sciences (pp. 1-8). Evanston, IL.

Anjaneyulu, K. S. R., & Anderson, J. R. (1992). The advantages of data flow diagrams for beginning programming. Proceedings of the Second International Conference on Intelligent Tutoring Systems, Montreal.

Corbett, A. T., Anderson, J. R., & Fincham, J. M. (1991). Menu selection vs. typing: Effects on learning in an intelligent programming tutor. Proceedings of the International Conference of the Learning Sciences (pp. 107-112). Evanston, IL.

Harvey, L., & Anderson, J. R. (in press). Transfer of declarative knowledge in complex information processing domains.

Wu, Q. (1992). Knowledge transfer among programming languages. (Doctoral dissertation, Pittsburgh, PA: Carnegie Mellon University).

Wu, Q., & Anderson, J. R. (1991). Knowledge transfer among programming languages. Proceedings of the 13th Annual Conference of the Cognitive Science Society (pp. 376-381).

Wu, Q., & Anderson, J. R. (1993). Strategy choice and change in programming. International Journal of Man and Machine Studies, 39, 579-598.