TEACHING COMPUTER PROGRAMMING IN INTERMEDIATE SCHOOLS

A thesis presented in partial
fulfilment of the requirements
for the degree of Doctor of Philosophy
in Education
at Massey University

Richard John Spence

1975

Abstract

This investigation concerned classroom learning of a computer programming course by Form 2 pupils in New Zealand Intermediate schools. Samples were employed representing the full range of ability levels found in such schools.

The programming task was divided into a pre-coding phase and a coding phase, and the capacity to perform the tasks relating to each of the two phases were postulated as separate abilities. This division was shown to be justified. Nevertheless, measures of the two abilities were found to be moderately correlated, and each also correlated moderately with a measure of mathematical attainment. Analysis of the results showed that these correlations were not due to general intelligence alone. The fine structure underlying the relationships was also examined.

In the study, it was further shown that three measures of academic achievement predicted attainment in the programming course more effectively than fourteen personality measures. Some similarities and some differences were discovered between the results of this prediction study and similar studies with adults.

Finally, two different teaching sequences were compared against each other and with a control group. It was established that mastery of the pre-coding phase of programming was improved by teaching, but that the place in the course where this teaching was given made no significant difference to end-of-course achievement. On the other hand, altering the timing of the instruction in the elements of the programming language was found to produce a significant difference in mastery of coding skills.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Acknowledgements

The writer thanks his supervisors, Professor C.G.N. Hill, Dr D.M. McAlpine and Dr R.E. Sass of Massey University's Department of Education for their help. Their valuable suggestions and patient reading of earlier drafts of this thesis are especially appreciated.

Thanks are also due to Mr J. Dowling, formerly District Senior Inspector of Schools for encouraging the project, and to the Principals of the three participating schools: Messrs R. Ward, J. Morris and P. McCarthy.

To those others in industrial, academic and educational circles who have made varied contributions, the writer acknowledges profound indebtedness.

PREFACE

The burgeoning growth of computer studies in primary and secondary schools makes research into its educational significance a matter of urgency. The growth has been accelerated by the falling cost of computer hardware and the increasing number of teachers, particularly of mathematics, who have some computing in their background and a desire to teach it. Since the United States is the world's major manufacturer of computing equipment, it is there that developments have been most spectacular: the Survey of Computing Activities in Secondary Schools (Darby et al., 1970) gives a detailed account of the extent and character of the growth in that country. But the growth has also been described by the OECD Centre for Educational Research and Innovation as occurring

> "... on a haphazard and random basis. In the main, the impetus has come from individuals, many of whom were interested in only one facet, for example, the use of computers in the teaching of mathematics. One result of this is that in too many schools, computer education is restricted to mathematics lessons or even to mathematically-gifted pupils. When this happens, many of the most valuable rewards of computer education are lost." (OECD, 1971)

Well-founded research will enable educators to decide whether or not computer education in schools really would produce worthwhile outcomes. Certainly, programming enjoys widespread support as a subject for study in schools: The quotations that follow are typical of almost daily utterances favouring the introduction of programming into the school curriculum:

> "In the preparation of a program, logical thought, care, and precision are required ... The discipline of systematic thinking and clear communication that are associated with the logical aspect of computer work are themselves of educational value." (Scottish Ed. Dept., undated)

> "The disciplines of problem definition, systems analysis and design, flowcharting and programming have been shown to significantly develop the child's ability to approach situations in such a confident, ordered, and creative way."
> (ICL/CES, 1972)

> "... I propose creating an environment in which the child will become highly involved in experiences of a kind to provide rich soil for the growth of institutions and concepts for dealing with thinking, playing and so on (through computer programming) ... the empirical evidence is very strong that we can do it ..."
> (Papert , 1971)

Statements like the above share a common characteristic: belief in an undefined "thinking" skill and an unspecified mechanism by which computer programming activities stimulate it. Because the advocates of programming as a school subject have so far concerned themselves with demonstrating that programming <u>can</u> be taught at quite junior grade levels, little has been done to investigate the proper place of programming in the school curriculum. At the same time, little is known empirically about how best to teach programming, and to whom. Cox (1972) has written:

"In a recent search of the literature it was surprising and disconcerting to find little on the effectiveness of teaching programming. Most studies concentrated on the tools used rather than on the problem, though Sackman et al. (1968) document what any manager knows: the gulf between the good and the mediocre programmer .... it is high time we found answers to some of the questions by controlled experiments with a wide variety of groups."

A main aim of the present study was to provide some of those answers.

Explanation of Technical Terms

Studies of computer education cross the boundaries of Education and Computer Science, and therefore draw on the terminologies of both. Because computer terms will be used throughout this account, and because they are essential to the discourse, the following explanation is provided at this point rather than in an appendix:

An ELECTRONIC DIGITAL COMPUTER is an assembly of interconnected devices, mainly electronic but also electromechanical, capable of performing a wide variety of functions on information that is presented in the form of CHARACTERS (letters, digits, punctuation marks). The business of getting information into the computer is called INPUT, and the same word is used without ambiguity to denote the ingoing information itself. (The term DATA is strictly equivalent to "information", but is most usually encountered in relation to input.) Once information has been input, the computer can PROCESS it and ultimately OUTPUT the modified information. The processing phase includes all the data-manipulation (e.g. arithmetic), logical tests, and management of the computer's memory, or STORE, where information is held while it is being worked on. By means of INSTRUCTIONS to the computer, a human can control the input, processing and output that occurs.

A set of instructions to perform a specific function is called a PROGRAM. However, as it can modify the data in its memory, the computer can also modify its program; it is because of this distinctive feature that modern digital computers are sometimes called STORED-PROGRAM computers. Because the program is stored, its sequence can be altered by special BRANCH instructions incorporated in the program itself. An important use of branch instructions is to create LOOPS in programs whereby certain sequences of instructions are operated repeatedly. Programs will also be self-modifying if they contain CONDITIONAL INSTRUCTIONS

which the computer only obeys under certain specified conditions. All non-trivial programs use conditional, branch, and conditional branch instructions.

PROGRAMMING[1] involves converting a problem into a set of directions to a computer to solve it. The function is sometimes broken down into several parts, particularly if the problem is very complex. A specific procedure for solving a problem is an ALGORITHM[1]. The process of writing the detailed step-by-step instructions for the computer to follow is CODING[1]. After a program is written, it is tested by letting it perform its function on test data to which the proper solution is known. This process is PROGRAM CHECKOUT or DEBUGGING; when it is done using pencil and paper rather than the computer itself, it is called a DESK-CHECK.

Careful and complete algorithm design and desk-checking must be carried out in advance if a program is to be run on a BATCH-PROCESSING system - one in which completed programs and data must be batched together and input to the computer at one time. Batch-processing, especially when one batch contains numerous programs, is cheap and efficient in computer terms but necessarily involves some delay between preparation of a program and receipt of the resulting output. The delay may sometimes be as little as an hour (e.g. for a high-priority user in a university computer unit), but is commonly a day or more. In contrast to batch-processing, CONVERSATIONAL PROGRAMMING gives the programmer direct access to the "live" computer system via an interactive terminal, enabling him to get instant results even from incomplete programs.

The programmer will be expected to produce some descriptions of his program and how it operates so that others may understand how it works. This DOCUMENTATION may include a FLOWCHART[1]: a graphic description or diagram of the various paths and branches followed by the program.

---

[1] This important term is defined in detail in Chapter II

The repertory of instructions available to the programmer for a specific computer is that computer's MACHINE LANGUAGE. (Because this use of the word "language" is somewhat misleading, human languages such as English are distinguished as NATURAL LANGUAGES.) HIGHER-LEVEL LANGUAGES have been developed to help the programmer by simplifying the tedious aspects of writing machine language; these include FORTRAN and ALGOL for scientific work, COBOL for business data processing, and dozens of others each with its own special uses and characteristics. Generally speaking, the higher-level languages can be used on a wide range of models and makes of computers, provided appropriate COMPILERS are available. A compiler is a master-program that converts a higher-level language into machine language; programs that perform similar functions at a much simpler level are ASSEMBLERS.

SOFTWARE is the term used to denote the totality of programs, documentation and procedures required in order to use a computer; sometimes it is used more specifically to mean those programs of general usefulness (such as compilers) that are available to all users. Software is contrasted to HARDWARE - the physical machinery itself, which is controlled not by the programmer but by a COMPUTER OPERATOR, whose job includes such matters as inputting the computer's work and collecting the output to be returned to the user. Though in real-life computing it is unusual for an operator to engage in programming, he does make use of a master item of software called the OPERATING SYSTEM that helps him in sequencing jobs, accounting, and calling up other software.