CURRENT TRENDS IN PROGRAMMING

Peter C. Poole
Department of Computer Science
University of Melbourne, Melbourne, Australia

1. INTRODUCTION

In preparing these lectures for delivery to an audience composed primarily of physicists, I have had to make certain assumptions about backgrounds in computing. I have assumed that each participant is familiar with at least one high level programming language, for example, FORTRAN and has written a number of programs in that language. I recognize that some participants will have had much more experience with computers than this and that some sections of the presentation will be considered by such people as a little elementary. However, I suspect that the assumption is true for the greater majority and I will therefore proceed on that basis.

Programming is the art of instructing a high speed electronic digital computer about how it is to solve a particular problem. It encompasses specifying the problem in an appropriate form, designing the program, constructing the code, testing it to demonstrate that it does what it is supposed to do and debugging it to find out why it does not. As a technique, it has advanced considerably since the early days of computing. The first programmers, who were also usually the end users, often physicists and other scientists, had to program in machine code and exercise considerable skill and ingenuity to pack their programs into the small memories then available. It was soon realised that the machine itself could be used to carry out some of the housekeeping associated with programming to make the writing of programs a more convenient task for people. One of the first machines in Australia installed in the Physics Department at the University of Sydney in the mid 50's, came supplied with a primitive assembler, referred to as the "decimal order input". It was a great step forward since, although operation
codes still had to be expressed in hexadecimal, addresses could be in decimal and, what is more, relative to base addresses defined at various points in the program. Thus, code could be conveniently rearranged in memory without requiring any modification. Such refinements as symbolic operation codes and locations, literals, conditional assembly, macros and the like that one expects to find in a modern assembler, were absent. Documentation consisted of writing on a listing of the program since there was no provision for incorporating comments directly in the text. In spite of such primitive facilities, some remarkably effective programs were constructed and operated successfully for many years. As the machine only had 1K 40-bit words and a 75 microsecond add-time, considerable attention had to be paid to the problems of efficiency which took precedence over the other qualities we might expect software to possess today.

The invention of high level languages such as FORTRAN and COBOL, in the early fifties, saw a further improvement in the power available to programmers who could now write in languages akin to natural language and mathematics, relying on the computer to translate their program into its internal codes. The use of computers boomed. No longer considered just esoteric devices for scientific calculations, they found wide spread application in business and industry. The number of people required to program this vast array of machines also increased, almost exponentially. The professional programmer was born - someone who took the specification of a problem, constructed a program to solve it and has little interest in using the results once he believed them to be correct. Such people would often move on to new tasks as soon as the program was put into service leaving maintenance and enhancement to others. This required the establishment of new standards for documentation and program clarity.

In the decade from the mid-fifties, there were many advances in the art of constructing software. Most of the programming techniques we will use today were invented during this period. The first computers had no operating systems and very little support software. Although the ubiquitous memory dump was known even then, programmers debugged their programs on the machine by examining and changing memory locations, setting address traps, monitoring sense switches etc. They usually had to patch their programs in absolute binary. With the development of the high level languages came the introduction of monitor systems which ultimately led to the operating systems in use today. It seemed at the beginning of the 60's that the golden age of programming had dawned. New languages were appearing with great frequency providing programmers with powerful constructs for expressing their algorithms. Compiler writing had become almost mechanised. Operating systems supplied convenient environments for program check-out with traces, snapshots, tracebacks, postmortem dumps and interactive debuggers. Encouraged by the success of the early time-sharing systems, the major manufacturers were