SOFTWARE: WHO ARE THE AUTHORS?

• *by Andrew McBurnie*

Preface

The difficulties being experienced by the courts in keeping up with the ever increasing technicalities of computer law can be seen in a recent decision of Northrop J in the Federal Court of Australia in *Autodesk Inc & Anor v Martin Peter Dyason & or*. This is the first decision to look at the definition of "computer program" within the meaning of the *Copyright Act* 1968. The case, discussed in the opening article this issue, higlights some of the difficulties which are referred to in this article.

Introduction

The growth of computer technology has seen the wide-scale commercial use of a type of artifact which has commercial value almost solely by reason of its intellectual content, and very little by reason of its physical production cost. For commercial exchange, this artifact nowadays exists in the intangible form of electrical or magnetic patterns – the computer program.

The first primitive computer programs were produced over one hundred years ago by the Countess Ada Lovelace, working on the mechanical computer of Charles Babbage. It is only in the last twenty years that the wide-scale use of computers has created a mass market for computer software, with attendant problems in the protection of the investment used to produce the software.

The computer program, as purchased on diskette, or some other form of magnetic media, is the final manifestation of a complex process of specification and authorship. The nature of this process raises issues relating to the protection of software which may be of interest to the legal profession.

Levels of Specification

A computer program, running in a computer system, is the end result; the lowest level of expression, of a series of higher level "programs", or specifications. Major stages in the specification and design of the system are often performed by people who do not actually write the program. The development of a computer program may be done by program designers, or "software architects", who produce algorithms, data structures and other documents which they describe to the programmers in various ways, for example module diagrams, flowcharts, screen layouts and a form called "pseudo-English" or "structured English". This last form of description is also commonly found in computer science journals, as a way of describing new algorithms.

In the development of a large, complex computer program, or a suite of programs, there is a descending hierarchy of expressions of an idea. Each level of expression is more limited, becoming more mechanical, down to the unambiguous and formal precision of the programming language itself. Below that level is the object code and run time code, and finally the invocation of the run time code whenever it is actually used in a computer.

Levels of Authorship

A compiler for a particular programming language does not merely convert a program written in that language into the internal language of the particular system for which the program is targeted. It also puts a great deal of extra code into the converted program, (the object code). This extra software is commonly referred to as a "run–time" system, and is usually supplied by the designers and programmers of the compiler.

For example, on computers which have no multiply or divide instructions, the compiler may generate multiplication and division routines built from complex iterations of addition and subtraction respectively. Similarly for calculations involving more complex functions such as square roots and trigonometric or hyperbolic functions.

More commonly, for computer programs which perform complex manipulation of data records, complete suites of code providing means of indexing those records will be included. Methods of generating screen displays or communicating with other computer systems may also be generated, if the program calls upon them.

For some years now, most commercial compilers have simply included the "names" of the particular run–time units which are needed, and these are then separately copied into the final program by a piece of software commonly known as a "link editor", or just a "linker". The linker is also used to combine the efforts of several programmers, each working on a module of a large program. With some operating systems, the needed run–time units can be loaded automatically into memory whenever the program is executed by a user. This saves on disc storage, since many programs may all use the same run–time software.

Sometimes, run–time facilities are sold as distinct products, either as source code or as object code, for other software producers to incorporate into their own works. In the case of both compiler run–time systems and separate products, advertisements in the trade press will frequently promise that no royalties will be claimed by the authors of these "included" products.

On a similar theme, when a computer program is loaded into a computer system for execution and even *after* the inclusion of run-time units, many of the instructions the running program will issue to the computer, including those of any "included" code like run–time units, will not be actual machine instructions defined for the hardware of the machine. They are requests to the operating system software of the computer for certain operations to be performed. Typically, these are low–level input and output operations to devices such as disc drives, tape drives, printers and computer screens.

These services are provided by the authors of the operating system, and the authors of the application code, at all levels, rely on the correct functioning of these services. The variety of authors involved in the creation of software for a particular computer, may often explain that computer's occasional schizophrenic behaviour.

Levels of Ownership

Just as ownership resides in computer programs, including programs which are language compilers, a form of ownership must lie in some programming languages themselves, independent of the computer programs which implement those languages. The language called "COBOL" is one example. It was defined in the early sixties by a group of computer manufacturers working with the United States Department of Defence. Any COBOL manual for a particular compiler must include an acknowledgement of the various forms of ownership involved, if the author of the compiler wishes to call his implementation "COBOL". The intent of this only becomes clear with the much more modern language "Ada"

(Continued page 12))

Notice is Given of the

ANNUAL GENERAL MEETING

of the NSW Society for Computers and the Law

on

Wednesday 6 December 1989 at 5.00 p m

at

Level 2, Law Society
170 Phillip Street
Sydney

All members are welcome

(Software.... continued)

(named after the Countess Lovelace).

Sammet (1986) gives a good description. The United States Department of Defence is intent upon preserving its future investment in software by ensuring compatibility across all its widely differing computer systems and software. Previous languages, including COBOL, have shown a tendency to split off into various dialects in a manner somewhat analogous to human languages. This reduces the portability across different computers of programs written ostensibly in the same language.

The Department of Defence has in recent years trademarked a language called "Ada", and has written a suite of standard Ada software which Ada compilers must handle successfully in order to be allowed to be called "Ada" compilers. It has specifically forbidden subsets or supersets of Ada and has established a formal liaison process with the American National Standards Institute to control the development of the Ada language specification.

Thus, Ada is the property of the United States Department of Defence, while individual Ada compilers are the property of their authors, and individual programs written in Ada are the property of *their* authors. This is another hierarchy of ownership, which parallels that described above for the general development of software.

The Vanishing Programmer

Some of the latest programming languages, such as PROLOG, no longer require the programmer to specify in step by step form *how* a task is to be carried out. The programmer simply specifies in great detail the *objectives*, by means of stating rules in a restricted type of formal logic. The sequence of statements in a PROLOG program often have the same effect in whatever sequence they are written.

The interpretation of PROLOG "rules" to achieve the objectives of the program is carried out by a low level piece of software called an "inference engine". The inference engine expresses the ideas embodied by the rules contained in the particular PROLOG program which is being executed. The process actually carried out by the computer which is running a PROLOG program might be regarded as a type of cooperation, at a distance, between the authors of a set of PROLOG rules, and the authors of the inference engine. Who, then, is the creator?

The inference engine will have been written in a conventional programming language, and hence carries all the attributes of software as described above.

At least one supplier of a PROLOG environment also makes available complete suites of predefined PROLOG "rules", which software developers can incorporate into their own PROLOG programs, in a manner similar to the more conventional run–time software described above.

Comment

It seems that it will be increasingly difficult for the law to keep up with the interlocking complexities outlined above, and the pace of new developments in the technology. Indeed it is becoming increasingly difficult for computer professionals to unravel the maze of authors involved in software development.

The manifestation of a computer program will change from year to year as the technology changes. The same program may be written several times, and in different languages for different computers and different operating systems.

It is traditional human documentation which seems to be well catered for by existing law. From a computer professional's point of view I suggest that the machine implementation should be regarded at most levels, as mechanical and legally trivial. If what is to be protected is the expression of an idea; (which may be arguable after Autodesk Inc.) with the result that programs are to be protected, then perhaps the primary form requiring protection should be the highest level of expression. That is, the level at which other intellectual works have previously been protected – as conventional tangible items of technical human communication: charts, English language descriptions (even of an obtuse form such as "structured" English). These documents are all specifications, on whatever medium they are held, and the resulting operational computer program is a lower level manifestation of that specification.

The sheer complexity of the technical problems concerning authorship and ownership means that attempts to isolate a protectable unit solely in the form of program source code, object code, run time binaries or any other merely mechanical expression of an idea, are likely to become bogged in a mire of argument.

During the development of software, the owners should always formally establish the connection between their higher level of expression, that is all the hierarchy of their design documents, and the program (or programs) written to that specification – the next lowest level of expression, and then as far down as they are compelled to go. Clear and concise documentation is necessary for good software engineering.

Thus, the design and specification documents are concrete expressions of ideas which can be protected as intellectual property. The major task should be to show an unbroken succession from the higher levels of specification to the lowest, and to be able to assert that the lower level manifestations are the same thing in another form.

References

Sammut (1986), "Why Ada is not just another programming language", *Communications of the ACM*, August 1986, p722-732.

• *Director of Computing Services, Blake Dawson Waldron, Solicitors.*

WESTERN AUSTRALIAN SOCIETY

The Western Australian Society for Computers and the Law is operational again, with a new committee and some upcoming activities planned. On 15 November a seminar on "Laptops for Lawyers" will be held. For details of planned activities, and any other enquiries, contact either:

The President: Kevin O'Toole
(09) 221–4748
or the Secretary: Michael Pattison
(09) 322–0321