SOFTWARE PROTECTION: HOW MUCH AND WHAT KIND?

M E Lesk

Rapporteurs: Jim Smith

Software protection: how much and what kind?

Michael Lesk

Bellcore
445 South St
Morristown, NJ 07960
USA

ABSTRACT

Intellectual property protection rules are having a major effect on how computer scientists work and what happens to their output. There is a danger of people being forced to extreme strategies such as deliberately ignoring all prior work in writing code, or introducing trivial variations in the appearance of all programs, as a way of getting around legal problems. It is important for the future of software that we at least understand the situation, and if possible improve it.

1. Intellectual Property Law Goals

The intent of laws regarding intellectual property is to encourage people to invent things and advance technology, by assuring them that if they do so they can be rewarded. It is widely believed that investors will not support work that can not be protected in some legal way, since otherwise they can not recover the development costs.

Most software companies do not rely heavily on patents. Microsoft, Novell, Oracle, and so on mostly appear in patent court when somebody sues them. The following table shows the number of patents dated 1994 from various companies:

Company	Patents
IBM	1362
Hitachi	1192
GE	993
Fujitsu	615
Microsoft	27
Borland	3
Lotus	0

An exception, a very patent-dependent software company, is RSA (which controls, for a few more years, the encryption patents).

Some companies have in fact done very well out of patents. Texas Instruments, for example, has boasted that their patent licensing brings in more money some years than the total profits of the company (since this implies they lose money building things, I'm not quite sure why they are so proud of it). Refac is a company which does nothing but litigate. They find apparently valuable patents, sue the infringers, and split the receipts with the patent holders. For example, there is a patent known as the Pardo patent, US 4.398,249. It appears to be a patent on spreadsheets, applied for in 1970 and granted in 1983. Refac bought this patent and has now sued Lotus, Microsoft, and Ashton-Tate.

I am not a lawyer, but let me start by reviewing some basic principles of how one can protect programs. I shall be talking primarily about United States rules, but there is a trend towards harmonization of intellectual property rules, partly because of political pressure from the U.S. GATT has recommended that other countries adopt copyright protection for computer programs, and most are following this path. This is part of a long tradition by which developed countries attempt to protect intellectual property and less-developed countries attempt to make it available easily. In the 19th century the United States was known for publishing books from England without paying royalties to authors. And in the 18th century, Ireland was known for pirate editions. Interestingly, when the US agreed to pay royalties to foreign authors in the 1890s, the reason was not pressure from foreign countries, but pressure from American authors who argued that US publishers were preferring to publish the works of foreigners since they were cheaper.

There are, of course, many existing industries, with different kinds of intellectual property law. The book, photography and magazine industries rely on copyright law. The drug industry, as an example, relies on patent law. Other industries operate without any intellectual property protection at all. Dress designs, for example, can be imitated at will; this is why Vogue Patterns and similar publications are allowed to publish patterns reflecting the work of Paris couturiers. Architectural designs can be imitated, as can business methods.

Certain kinds of outside force have required other adjustments to the law. During World War I, for example, the U S airplane patents were divided between the Wright brothers and Glenn Curtiss. They refused to license each other, and as a result nobody in the United States could build a state-of-the-art airplane. This lasted until the U. S. entered the war, at which point the Army realized that American pilots would be shot down by Germans who were flying better planes because they were not required to obey U S patent law. The Navy imposed a cross license on the owners of airplane patents.

An test case is the typefont industry. In the United States, typefont designs are not protectable. The names can be trademarked, so that you sometimes find fonts which look just like Helvetica (a trademark) being sold under names like Swiss or Geneva. But the design itself can be copied (and there is at least one US company that has made a living copying Mergenthaler and other fonts). In Germany and France, on the other hand, font designs can be legally protected. If the standard arguments of the lawyers are correct (without legal protection nobody will invest in innovation) one would expect that advances in type design would be made in Europe. In fact, for some years most type design advances are made in the United States, since they are tied to the advent of laser printers and Postscript, both of which are US innovations. In fairness there are sufficiently few internationally famous type designers that it is hard to get away from describing the situation with phrases like ''Frederic Goudy was an American, Hermann Zapf was a German, Stanley Morison was British, and Charles Bigelow is another American.'' But there is certainly no evidence that the industry has moved to Europe for lack of intellectual property law in the United States.

2. Intellectual Property Protection methods

To summarize briefly, there are three basic ways of doing this protection. These are trade secrecy, copyright, and patents.

The easiest to understand is trade secrecy; you simply don't tell anyone what you are doing, or make anyone you do tell promise not to tell anyone else. This has some practical limitations, particularly for widely distributed software systems. The appearance of a user interface, for example, can hardly be declared to be a trade secret if the program is widely used. Secrecy also does not support the public goals of advancing the general state of technology well, since the technology involved is not disclosed to anyone. Also, the more the technology is used, the greater the chance that through some error, the technology will become known to somebody who hasn't made any promises to keep it secret. Such a person can freely redistribute the information. In short, trade secrecy can only be used for something that can credibly be kept secret, e.g. a process used only within the company rather than a product sold to outsiders. Trade secrecy has no explicit time limit, but nobody should be too optimistic about how long a particular recipe can be kept secret. Perhaps the best known trade secret is the formula for Coca-Cola, which has been kept secret for about a century.

Copyright protects a written text. It does not protect an idea; you can not copyright the idea of ''man goes on whaling voyage,'' but you can copyright the text of *Moby Dick*. Copyright can protect more than just a text, however: images or sounds can be copyrighted (and are, routinely). The first copyright law was in 1709. Before then, authors and composers were routinely ripped off; in the 19th century musicians sometimes went to a concert, and wrote down the music so they could perform it later themselves.

Note that copyright only protects a text. If somebody else can look at your product, and simply write a clone using different code, they in general can ignore the copyright in your text. It does not protect the functionality of the code. New kinds of work come under copyright only slowly. Audio recordings, as opposed to musical scores, fell under a legal deposit requirement in the United States more than fifty years after they became common. The same was true for movies; in the early years of motion pictures they were protected by depositing prints of each frame of the movie.

Copyright is regulated in most countries by the Berne convention, which provides that an expression of an idea is copyrighted automatically upon creation. Registration with a national authority is not necessary (although it is often done as a simple way of establishing the date and to aid in damage claims). Many Americans still remember some old rules in which copyrighted items had to be labeled with the copyright notice and publication without that forfeited the copyright. That is no longer true.

The time period of copyright is life of author plus 50 years in most of the world, or 100 years for works written by corporate employees. In the United States it is currently 75 years from date of publication (replacing an earlier law which provided 28 years plus a single renewal for another 28). The U. S. will switch to the Berne rules in the next century, and all of these time periods are being lengthened by 20 years (so that U S copyright will run 95 years and Berne copyrights life plus 70 years). For practical purposes, in software the copyright period is infinitely long.

The administrative cost in getting a copyright is minimal, but the difficulties of dealing with copyrights afterwards can be serious. It can be hard to find out who has a copyright, owners can have different and perhaps irrational views of how to sell their rights, and the process can be very slow. For example, IBM produced a CD-ROM to commemorate the Columbus voyage of 1492. It is said they spent over $1 million clearing rights, of which only $10,000 was paid to rights holders; the rest went to administrative costs.

Patents, by contrast to copyright, can protect an idea. Finally, patents. The first British patent law dates from the late 17th century. Patents give the broadest protection, since they protect an idea. A patent, in software, can protect an algorithm. There had been considerable argument in the United States about the patentability of software (since the original law talks about devices and processes). After a long discussion of player pianos (which were copyrighted and not patented) and of accounting formsheets (which were not protectable at all) the US Supreme Court first said software was NOT patentable (Gottschalk v. Benson, can't patent a pure algorithm) and then in 1981 the court succumbed to the pleadings of many large companies and agreed to the patentability of software (Diamond v. Diehr). You do have to have an application--you can't patent a pure algorithm. A patent until recently in the United States lasted 17 years from date of issue; now it lasts 20 years from date of application. Patents require disclosure of the invention, although there is an art in writing such documents as to be of very little use to anyone trying to duplicate the method. Patents cost about $10-25K for the legal costs and the infringement suits are $100K and up.

There are some other forms of intellectual property, such as design patents and the special protection for chip masks, but these are less commonly used. There is also the trademark law, which protects only the name of the product, and various rules reflecting ''trade dress'' which restrict people from pretending to be somebody else. To return to Coca-Cola, they have a trademark on the name; many years of litigation determined that Pepsi-Cola was sufficiently different to be acceptable while Taki-Cola was too close to be allowed. So Coca-Cola, to continue the example, has a trade secret formula, a trademarked name, copyrighted advertisemens, and patents on things like ''apparatus for producing carbonated water'' (a machine which simultaneously mixes carbon dioxide with water and chills it).

3. Disputes in Protection Law

Traditionally, copyright law has protected creative expression, of the sort associated with authors of literature or composers of music. There are some rough edges relating to constrained expression and to data compilations. First of all, if there is only one way to say something, copyright law will not protect it. This has come up in discussions about imitating microcode on different computers. Companies have successfully gained the right to use similar microcode by claiming it is the only way to do the task. This is commonly tested by having a development group work in isolation of the microcode they are trying to reproduce; if they come up with the same solutions to the same coding problem, the courts will allow it.

Another important variation is whether or not one can copyright data compilations. For many years in the United States a rule known as ''sweat of the brow'' was enforced, saying that if people had spent enough effort collecting something they could get protection for the result. In 1991, however, the Supreme Court ruled in a case called Feist Publications v. Rural Telephone Services that simple effort was not enough, that the law required creativity and that alphabetizing the names in a phone book did not represent creativity. The Court is willing to be very generous about interpreting creativity, so that almost any kind of arrangement will provide protection, but phone books didn't qualify. Among the later cases are one in which the selection and arrangement of baseball box scores for playing some kind of predictive baseball games (Kregos vs. Associated Press) was held to be creative while the publication of horse racing charts was held not to be creative and thus not protectable (Lalli Enterprises vs. Big Red Apple), Although an alphabetized ''white pages'' phone book is not protectable, a ''yellow pages'' type phone book is (BellSouth v. R. R. Donnelly and Key Publications vs. Chinatown Today).

In Britain there are several variants on copyright which protect activities that don't normally rise to the level of creative literature. There is a protection for typographical arrangements, covering the appearance of a book whose text is not copyrighted (preventing people from photocopying it). There is also a ten-year data compilation copyright.

An important dispute in the U S is over the appearance of computer interfaces. The question is whether it is legitimate for a company to produce a program that looks the same and operates the same as another company's program, i.e. to write a clone. This has become known as the "look and feel" protection rule. This series of cases began with a strange set of minor cases involving individual programmers writing code for such groups as dental laboratories, but became of major economic significance when Lotus Development sued Paperback (VP-Planner) and Borland (Quattro) for cloning the Lotus-1-2-3 screen format. They won (the Lotus/Paperback case was then settled on appeal) and the U S law at the moment is that one has to make changes in the appearance of an interface when writing a program that does the same job. Thus you will notice that the trash can on the Macintosh does not look quite like the wastebasket on the HP windows system. Interface design, by the way, is also protectable by copyright in Europe.

A phrase from a law review article, talking about the ''look and feel'' of a program, has been used for some time to discuss interface copyrights. This phrase has not been adopted by the courts, however: an infringer must be shown to have copied specific features, not just the general impression of the program being cloned.

One rule that might have protected some coders was the parody exception. The courts have long recognized that few copyright holders would grant permission for a parody, and provided an exception for satire. Berkeley Software sells ''After Dark,'' one of the top screen-saver programs, featuring toasters flying across the computer screen. Delrina produced a screensaver program in which Opus the Penguin (a Berk Breathed cartoon character) shot down flying toasters. Berkeley sued, Delrina argued that their screen saver was a satire. The court rejected this on the grounds that it was a commercial operation rather than a traditional parody. He also suggested that it mattered whether the toasters were exactly the same design or not, again reinforcing the point that to infringe an interface copyright exact features must be copied.

Another legal question that is likely to affect software coders is legal liability. Those who distribute software that operates a machine or otherwise might cause damage are already aware of the problems. What is new is the question of liability for information distribution systems. Traditionally, book publishers are not liable for damages caused by false or misleading information in books. For example, decades ago people tried suing publishers about recipe books which gave bad advice for dealing with exotic plants or mushrooms (Cardozo v. True). They got nowhere. The warranty on a book is that the binding and paper will hold together, not that the content is correct. The publisher must also avoid libel and copyright infringement, but does not need to worry that someone will be misled by the text. In the U.S. there is only one contrary precedent: someone has successfully sued the publisher of an aeronautical chart, claiming that it contributed to a crash involving a small airplane.

What about information systems? Are they like books, or are they like stockbrokers giving advice, who are held to some kind of responsibility for what they say? So far, we don't really know. But there are some bad precedents. Microsoft Bookshelf, after all, has a part number, not an ISBN. Purchasers need more help with a computer system than a book, and may thus put a higher degree of responsibility on the seller. The libel standard still applies: Prodigy has been held liable, in a very publicized case, for critical comments made about a stockbroker on one of its bulletin boards by a user. Prodigy advertises that its bulletin boards are moderated to remove obscene language and that it is a ''family-friendly'' computer system; this caused the court to assign it responsibility for what the user had posted.

Realistically, in the present state of American law, it is likely that online information publishers will have to be more careful than book publishers about what they distribute; it is likely that within five years, some attorneys with a very sympathetic client (perhaps an elderly investor who has lost his or her life savings following the investment advice of a program which turns out to have been written by a 13-year old) will recover damages from the online operator or even the telephone company.

The patent area has produced even more disputes than the copyright law, because of US problems with software patenting. For many years patent examiners were not trained in computer science, causing them to have difficulty judging how novel ideas really were. In addition, with no history of software patents, the tendency for patents to cite primarily prior patents as references also made it hard to evaluate the novelty of some idea. As a result of the many software patents (over 1000 per year now) there are some which are clearly erroneous. People have patented line numbering, alphanumeric sorting, hidden line elimination, and many other previously known techniques.

Among recent disputes is one over an AT&T patent on ''backing store'' which claimed the idea of providing extra memory so that windows on a screen would be saved in memory even when not visible. The challengers to this were frustrated by the lack of publication in the early days of computer science (for example, we don't know who invented hashing). Many people claimed that the backing store technique had existed on one or another old machine, but were unable to document it.

Another dispute is over the LZW (Lempel-Ziv-Welch) compression algorithm; it came to prominence last December when Compuserve announced that Unisys was going to charge for the use of software that created GIF-format images, since LZW compression is part of GIF (and also part of the Unix ''compress'' command). After some public outcry Unisys has said they will only expect payment from commercial software products brought to market after 1994 (no payment for old programs or for those written for one's own use or as freeware).

The best known doubtful patent is the Compton's NewMedia multimedia patent, which appeared to control an astonishing range of multimedia display procedures (perhaps anything that linked sound, text and images on CD-ROM). A protest from many companies caused the Patent Office to open a re-examination of that patent, and it will probably be disallowed.

Finally, there are patent problems arising from the length of time it takes patents to issue. Patents in progress are like ''time bombs;'' they are secret, and a company can be unknowingly infringing, but when the patents issue they become valid. In more familiar patent areas such as chemicals it is rare for patents to take a long time to issue, so this is not much of a problem. In software some patents take years, and in fact some applicants have found that by filing amendments to patents they can delay the issuance of a patent, thus extending its legal life. The change from the old lifetime, counting 17 years after date of issue, to the new lifetime, counting 20 years from date of application, will help this somewhat.

In general, the patent process has had a rough time adapting to software. Most software, in reality, is not patentable; and it has been hard to establish what is patentable and how much should be covered.

4. What Should be Protected?

The key question is what kind of protection regime we would like to have for software, and what would be good for the industry. I do not see that when we introduced lawyers into computing we saw a significant improvement in innovation. Observation is that the main defenders of protection are large companies, despite the many statements that individual entrepreneurs will benefit most from software protection. There are some small businesses which have benefited, but again it is not always by building a business. For example, Hypercard is much better known than Zoomracks; Zoomracks has a patent on putting cards on screens, however, and collects royalties from Apple. There are certainly some large businesses (Sun and Adobe, for example) which oppose software patents.

An argument can be made that investors would prefer not to have legal protection. The reason is that they don't like the risk of a patent infringement suit. The current legal realm encourages people to get patents, keep quiet about them, and just wait for an infringer who can be sued. Encouraging people to design different interfaces just to be different is not helping technology and does not make products easier to use.

On the other side, software piracy is a very serious problem. The Business Software Alliance claims piracy losses in 1994 were $15 billion (fifteen thousand million dollars). Software piracy is above 95% (i.e. only one in twenty copies of a program is legal) in countries such as China, Russia, Pakistan, Phillipines, Thailand and Indonesia.

Some of this is being addressed through international negotiation, although the idea that a factory in China which had an order for 5 million fake holograms for labeling software copies was only fined $5,000 as punishment seems a bit weak. The issue is whether, in such an environment, technology development is discouraged because people afraid they can not recover any costs from their technology work simply don't do it.

Prof. Pamela Samuelson and others have argued strongly for a new kind of protection for software. They suggest that software is unusual in that it is a "cumulative" technology in which people build on each other's work, and that it is neither really a device or nor a text. As a result they propose that the legislatures create a new kind of protection for software, which they propose as fairly limited. It would have a relatively short term, with low administrative costs, so that it would have to be a registration type system, not an examined type system. They would like to prevent cloning, at least for some period of time.

It is not entirely clear why software has the market properties that it does. Although it is easier to change than software, it appears to have a longer market life. It has huge economies of scale, meaning that the first entrant with a product should have an immense advantage over followers, and yet such companies as Visicalc are no longer around. It should be easily transmitted, and yet there the same program is often written many times. For lack of a history, there is little emotional certainty about what is morally proper and what is not.

What we should, as software engineers, try to understand, are questions about what should be protected, and how. For example, we would all probably agree that it is good for society and technology if new advances become public, rather than being kept secret. This suggests that some kind of protection is desirable, to discourage inventors from simply keeping everything secret.

To start with the easiest step, most people would agree that source code should be protectable. The inability to be sure that people will not copy or adapt code has meant that much code is kept secret and that effort is wasted on things like decompilation. Progress in technology is retarded by these difficulties. Code looks very much like something that should be copyrightable, and copyright is at least a low-cost procedure.

But even copyrighted code is not readily published. We have access to large libraries of freeware, but much more limited libraries of tools available in source form. Part of the problem is the difficulty of tracking use of source code, and of arranging to collect for its use. The software industry has been innovative in designing systems like per-user site licenses; it would be valuable if we could produce systems patterned perhaps on music licensing or the "public lending right" that allowed people to readily use existing material, and pay standard fees.

Appearance of an interface is a harder question. Yes, there is work involved in designing a good interface, and perhaps it should be protected by more than market presence. However, the concept of making it difficult for people to use each other's interfaces does not seem productive. Imagine if something like this were true with automobiles, and Ford had the copyright on putting the accelerator to the right of the brake pedal, requiring Toyota to put them the other way around. We already have too much trouble with people forming semi-religious attachments to particular pieces of software; encouraging this by requiring every single piece of software to develop a distinct interface does not seem productive. At least, interfaces by their nature are public. Again, if they are going to be protected, some kind of compulsory licensing might be productive.

Algorithms are one of the most fought-over areas. Should somebody who invents a new cryptographic technique be able to protect it? If so, for all applications, or only for some? Unfortunately here the system has granted some extremely general and bad patents, and it may be that some kind of narrower protection would be better. The patent system may not be appropriate for software, but right now there is nothing better. It may be that we can make do with copyright, stretching the definition of 'performance' to say that a new implementation of a copyrighted algorithm is an infringement under this view. Or it may be that we should argue for a new kind of protection, as suggested by Prof. Samuelson and others. Again, patents and copyrights have the advantage that the work at least becomes known.

Functionality is the other extreme from text. Can the general idea of what a program does be protected? Traditionally this has never been possible; patents protect a device, not all devices to achieve a single end. Similarly copyright does not protect general themes of literature or facts of history, only specific expressions of them. Does this mean that 'clone' software should simply be allowed? That may be the easiest answer; functionality is not concealable, so it will not interfere with technology advancement.

What could most usefully come from such discussions among computer scientists is probably not just advice to the lawyers, who are much more responsive to the business executives anyway. What we could best provide is some new methods for tracking the use of programs, administering some kind of rights system, and opening up some alternate ways in which people could publish what they are doing and then collect for it. Consider, for example, the way the Germans compensate composers for the copies of their work which are taped from radio broadcasts: there is a tax on blank tape, distributed to their composers. The public lending right payments in UK libraries are similar. Is there any similar method that would let us compensate program authors with low administrative costs, encouragement for publication, and a public sense of fairness? Could we come up with technology to encourage it?

5. Conclusions.

I have been going to computer conferences since 1969. At that time, I never heard anyone say that we had a problem because there were not enough lawyers in the area.

Most people would say that the single best idea created in computer interfaces is what is now called the Macintosh model (drag-and-drop icons). If ''imitation is the sincerest form of flattery,'' the progress of Microsoft Windows confirms the value of this model. It was created at a time when no one believed interfaces were protectable. And it was not created by either of the companies (Macintosh or Apple) that have made most of the money off of it; it was designed at Xerox PARC in the 1970s. For some of us, the ultimate irony was Apple suing Microsoft over a user interface whose basic ideas were not original with Apple, but taken from Xerox. In both cases, there were licenses covering some of the ideas, and the dispute was partly about contract law.

At least in the United States, there is a chance that there will be a revision of the copyright law, largely driven by the fears of publishers that digital information is too easy to copy and something has to be done to regulate the use of copyrighted information on the Internet (e.g. they would like to abolish fair use in the digital world). As computer scientists, we should try to think what changes if any we would like in the intellectual property rules that affect our work. What kind of legal protection methods will help or hinder the development of the field, the economic health of the companies involved, and the rewards given to programmers and scientists?

The history and development of this area do not suggest that patents have been very effective for anything; and that copyright in the familiar form (on the text of the code) is useful and workable. Copyright on interfaces, designs, and the like seems difficult to manage, and it is not clear that clone-writing needs to be discouraged. Perhaps we need to move back to less protection, rather than more. Our most useful contribution would be new technology to make it easier to track and pay for ideas as they are used.

References

1 Unix World's Open Computing, *Should Software Products Be Allowed Patent Protection?*, 11, p. 70, April 1994.

2 Pamela Samuelson, Randall Davis, Mitchell Kapor, and J. H. Reichman, ''A Manifesto Concerning the Legal Protection of Computer Programs,'' *Columbia Law Review*, vol. 94, pp. 2308-2341, 1994.

3 Brian Kahin, ''The Software Patent Crisis,'' *Technology Review*, pp. 52-58, April 1990.

DISCUSSION

Rapporteur: Jim Smith

Dr Herbert described how at APM, a certain word processor package was used and an appropriate number of licenses purchased for the team at release 3. Sometime later the team had expanded and the need arose to obtain more licenses, but by this time the producer was selling release 5 and would not grant further licenses for the earlier release. This then would be an example of a software company abusing its rights forcing APM into making unnecessarily large number of purchases of licenses for the later release, or attempting to make do with an inadequate system. Dr Lesk commented that the loss to APM might be the premature redundancy of the installed copies of the earlier release. This would not be an issue if a license was paid for on a "per time" basis. Dr Herbert noted a feature of distributed systems in that there is a choice as to whether a computation is performed locally or remotely, on the superhighway, and that the latter option is well suited to "pay as you go".

Professor Kopetz suggested that the problem with software patents might be a lack of experts in the patent office. In software, change is so rapid that it is more difficult to keep up than in other disciplines. Dr Lesk noted that the US patent office has a list of approved qualifications for staff entry and that until recently, computing science did not appear on this list. Mr Ainsworth expanded on the expertise required by pointing out that for an item to warrant protection it must not only be novel, but also be not obvious to a skilled practitioner, thus necessitating familiarity with a rapidly changing field.

During the talk Mr Simonyi noted the benefit of being able to copy pieces of software and wondered if there might be a place for a parallel in the software industry to the market in cheap out of copyright books. After the talk, Dr Herbert described an industry built on old versions of software, where a company may buy rights to an old version and compete with the original producer by producing an upgrade from the old version to a new version.

Professor Katzenelson asked for opinions on the work of Richard Stallman citing in particular the effect of the Gnu C compiler. While agreeing that Richard Stallman effectively destroyed all revenue for the Sun C compiler, Dr Lesk expressed doubt in the general applicability of the approach, claiming that Gnu offers a very low grade of support. He continued to agree with Professor Gladman's acknowledgement of the significant effect of the organisation but suggested that in the long term a low overhead technical solution to protection would be appropriate, particularly in the commercial sector.

In view of the difficulty of defining what sort of algorithm is patentable, Professor Gladman initiated some discussion on the question of patenting algorithms in UK. Professor Randell commented that genetic code information may be patentable. Dr Lesk commented that in the US there is special legislation for patenting ship design information and suggested that such an approach may be appropriate for software design too. Highlighting the uncertainty over algorithm patents, Professor Whitfield mentioned the case of the RSA algorithm which is patented in the US, but cannot be patented in the UK. Dr Lesk added that RSA is an example of a case in the US where a company exists purely on the strength of a collection of patents.

During the course of the talk, an example of an international dispute over patent rights was described where 5000 hologram copies were made in China and the fraud stopped with the imposition of a $5000 fine. With regard to the apparent small scale of the fine, Mr Simonyi clarified that the incident led to a threat of a trade war and was an example of the system successfully protecting the good guys. Professor Randell suggested that similar events might have taken place 100 years ago but with US read for China. Dr Lesk suggested that a distinction now is in the speed of distribution. Subsequently Mr

Simonyi pointed out that copying a hologram is a clear cut example of an offence. In the software industry however, experts cannot agree what should be patentable.

Ms Ringland wondered what are the high value items that need protection and suggested that source code itself is of limited value citing the success of a shareware industry as support. Dr Lesk inferred that the only value in a software system would then be in support. Ms Ringland continued saying that while a billion lines of code has some value, the real value is in the heads of the people who understand the software and can extend it. Mr Simonyi reported that, at Microsoft, protection from copying is viewed as necessary and that source code is accordingly kept in a locked vault.

Professor Gladman suggested that it may be appropriate, perhaps in the case of an algorithm such as RSA, for the ownership to go to a central body, but with some reward going to the creator. Dr Lesk noted that in the case of medical research ownership does go to the health authority, but not in other disciplines. Professor Gladman commented on the difficulty of predicting applications. One algorithm that is obviously patentable is that of quicksort, but algorithms for factoring large numbers considerably predated well known application. These considerations would appear to suggest a need for patenting ideas.