3 Faulty supplies

The first area we shall examine is what happens when there is something wrong with IT supplies. Nothing created by human beings is perfect, and that generalization is particularly pertinent to the software side of computing: it is a computing cliché that the "last bug" in a sizeable program is never located. What does the law have to say if something goes seriously wrong?

3.1 Breach of contract v. tort

First, we need to grasp a fundamental distinction between two ways in which "things can go wrong": *breach of contract*, and *tort*.

Suppose I am a car dealer and agree to sell you a low-mileage demonstration model, but after I deliver it you find that it is an old banger – someone else might have been happy to buy it, but only for a fraction of the price you paid. You will threaten to take me to court for breach of contract. We all know what a contract is: two parties promise to swap things they can provide and the other wants – commonly, though not necessarily, goods or services in one direction and money in the other. A contract for car purchase will include specific statements about the car, which have not been fulfilled.

But now suppose instead that I am pruning a tree that overhangs my boundary, and I do the work carelessly, so that a heavy bough falls on your new car parked in the road below and damages it. When you complain, you will not be very impressed if I blandly reply "Oh, that doesn't matter – we have no contract, I never promised to take care of your car"! Again you can take legal proceedings against me, but this time for a tort (French for "wrong"). I have done you harm in a way that I am not entitled to do, regardless of whether or not there was any prior relationship between us.

Both contract law and tort law are potentially relevant to IT supplies, and we shall consider each in turn. Under contract law we shall look first at some practical considerations facing a manager responsible for entering into computing contracts, and then at the chief issues concerning how such contracts are interpreted by courts. Under the "tort" heading there will be less to say. There are plenty of ways that unsatisfactory IT products may harm individuals outside any contractual relationship with the supplier; but we saw in chapter 2 that English law adapts to new phenomena through individual cases which establish precedents, and as yet there have been no significant cases about IT-related torts.

3.2 IT contracts

Managers who deal with contracts for IT supplies are often in a difficult situation. Many of them have a strong IT background, but sorting out contractual details is a whole separate ball game, and a difficult one. If, conversely, the manager has a business rather than IT background, his situation may be even worse: how can he foresee what technical points it is important to get down in black and white, if he does not really understand the technology too well? The situation is admirably summarized by Jeremy Holt, in a book which goes into more depth on these issues than the present book could aspire to:

> Pity the unfortunate manager. It has been bad enough trying to get the computer project organized. Now, possibly at the last moment, the contracts have arrived, some with print small enough to make the reader go blind. The manager suspects (rightly) that these contracts are one-sided in favour of the supplier, but knows that the project will only proceed if those contracts (or something similar) are signed. How does the manager work out what needs to be done and from whom advice can be obtained?[8]

IT contracts are difficult both because the law is complicated, and because IT is complicated. To quote Jeremy Holt again, "Among the most common causes of computer project failure are unclear client requirements and unrealistic client expectations."[9] A supplier company's sales representative will of course spend time discussing the client's needs and offering assurances about worries that the client voices (we are considering large-scale business-computing contracts here, not one-off purchases of a PC for home use); but the rep, and his employer, will be hoping that – if the client decides to go ahead – he will sign their standard contract terms. If the customer is willing to accept those, a great deal of expensive time and effort in sorting out the details of a tailor-made contract will be saved on both sides.

As an example of why that saving might be a false economy (for both sides), consider what is believed to have been the first occasion when a case turning on computer software came before a British court: *Mackenzie Patten & Co*. v. *British Olivetti Ltd* (1984). Mackenzie Patten were a firm of solicitors (so should have had more savvy about contracts than the average IT client!) They decided to computerize their accounts, at a period when it was still quite unusual for a non-technical business to use a computer. The Olivetti salesman discussed Mackenzie Patten's needs, and assured them that one of Olivetti's systems would be suitable. Mackenzie Patten leased it and spent considerable time trying to implement the intended functions, but it eventually turned out to be unusable for their specific purposes.

The problematic features were points which the written contract did not cover; so Olivetti may have thought they were in the clear. But in fact the judgement went in favour of Mackenzie Patten, because the salesman's assurances were treated as part of the contract. (Nothing in law says that a contract must be wholly written – indeed, legally it is quite possible to create a contract purely by word of mouth, though to enter into a significant business contract that way might be foolish, to say the least.) Olivetti had to repay the sums paid out by Mackenzie Patten, with interest. Meanwhile, from Mackenzie Patten's point of view the outcome was certainly better than losing the case – but they had wasted a great deal of expensive time and effort, and were presumably no closer to acquiring a system that would do what they needed.

In another similar case the plaintiff[10] could easily lose, perhaps because evidence about what the salesman said was contested and the judge did not accept the plaintiff's version. Sometimes a written contract will contain a so-called *entire agreement* clause, specifying that nothing external to the written document (such as salesmen's remarks) shall be treated as part of the contract – though a clause like that ought to be a signal to the client to make doubly sure that anything important said by the salesman gets written in. (In fact the contract in *Mackenzie Patten* did have an entire agreement clause, but for technical legal reasons the court treated it as inoperative.)

3.3 Letters of intent

With most things a business buys, their properties are understood well enough for the period between initial discussion and conclusion of a contract to be reasonably brief. An IT supplier, on the other hand, will often have to undertake a lengthy development project in consultation with the client, before it has a system ready to meet the client's needs, and both sides' understanding of those needs will be refined as the project proceeds. If the prospective supplier had to do that at its own risk, the expense might be difficult for its business to absorb and it would have an incentive to cut corners. The standard solution is a *letter of intent*: at an early stage in negotiations, the client puts on paper its intention to enter into a contract and agrees to pay for work done by the supplier in the interim – that way, the supplier can afford to make a proper job of exploring the client's needs and developing a suitable solution.

3.3.1 Service level agreements

Another general problem with IT contracts is that points which matter to the client are often details which would be "below the radar" of normal legal language. They need to be right, but they would not fit well within the kind of document a commercial contract is. In any commercial contract it is understood that the thing delivered has to be in saleable condition: one would not normally spell out explicitly that apples must not be rotten, a new car must go, or the like. But, with computer systems, the two sides may well have conflicting assumptions about what is saleable. Consider *Micron Computer Systems Ltd v. Wang (UK) Ltd* (1990). Micron claimed that the system it had bought from Wang was faulty, because it did not provide transaction logging. Wang responded that transaction logging was not part of the design specs of that system. On this aspect of the case, the judge sided with Wang and said that if Micron had needed transaction logging it should have made that clear. The essential problem here was that, for one side, mentioning this feature in the contract seemed as redundant as specifying in a car-purchase contract that the motor must run, the doors must lock, and so forth, but for the other side the feature was an optional extra.

The usual solution to this type of problem is a *service level agreement* (SLA): a separate document, referred to in the contract, but written by and for techie types rather than lawyers. An SLA will typically specify things such as technical quality standards, e.g. host/terminal response times, permissible levels of downtime, and so forth; and it will also lay down procedures for *change control*: in a sizeable development project it is certain in advance that specs will be modified in the light of experience as the project proceeds, so there must be agreed processes by which the client is kept up to date on progress and asked to consent to alterations of details. The SLA will lay down how particular departures from agreed service levels are to be compensated, for instance through adjustments to contract price. (The sanction of terminating the contract and claiming damages for breach of contract is an ultimate "nuclear option", not a first choice.)

Developing a useful SLA is itself a challenging task. The danger is that it can become an end in itself, full of metrics that can be objectively quantified but which have little to do with service quality as actually experienced by the client. There are recognised standards that can help. ITIL, the British government's IT Infrastructure Management Method, claims to be "the most widely accepted approach to IT service management in the world".[11] An international standard, ISO/IEC 20000, describes itself as "the first worldwide standard specifically aimed at IT Service Management" (and as "aligned with and complementary to the process approach defined within ITIL").[12] But these general standards are only guidelines; they cannot in themselves produce a suitable SLA for a particular contract.[13]

3.3.2 Governing law

There is no law requiring contracts executed in England to be governed by English law, though that is the default. Sometimes a contract will specify that if a dispute arises, it is to be resolved by a named private-sector arbitration service, such as IDRS or Longworth. Private arbitration has the large advantages of being cheaper and quicker than resolving a dispute in the public court system. For the client it also has a potential disadvantage, though. Arbitration proceedings are private, so the supplier under such a contract does not face the risk that a poor job will lead to adverse publicity. Negative publicity can cost a firm far more than compensating the client in an individual case, so it forms one of the strongest pressures on suppliers to do good work.

3.4 Interpretation of contracts

That is as much as we have space for on the practicalities of IT contracts. The other large issue is how a court will interpret the terms of a contract, if a dispute does arise (and assuming that the contract is governed in the normal way by English law).

Here we shall consider five areas:

- consequential loss
- goods v. services
- implied terms
- unfair terms
- development risk

3.4.1 Consequential loss

If a product (an IT system, or anything else) fails to perform as promised, the law will naturally require the supplier to refund the money actually paid for it. But the failure might have adverse knock-on effects on the purchaser's business. For instance, the purchaser could have been planning to bid for a piece of business which would have been lucrative if the bid was successful, and the failure of the product in question might make it impossible to bid for the work. That would be a very indirect effect (even if the purchaser had been able to put in a bid it might not have won the business), but it could be a serious one.

A supplier will want the contract to exclude liability for indirect (in legal language, *consequential*) loss. If the IT industry is to flourish, it is often reasonable that consequential liabilities should be excluded. IT products are often so general-purpose in nature that it is difficult to foresee the range of uses they might be put to (hence a supplier could not quantify the risk involved in liability for consequential losses); and potential losses will often be large relative to the value of an individual IT contract, so that suppliers could not easily afford to accept liability.

However, an IT supplier needs to appreciate that a court's view of which losses count as direct rather than consequential may be surprisingly broad. A leading case is *British Sugar plc* v. *NEI Power Projects Ltd* (1997–9). NEI supplied power equipment which proved defective, for a cost of about £100,000, under a contract which excluded liability for consequential losses. British Sugar claimed damages of over £5 million because the defective equipment increased their production costs and hence reduced their profits. The court agreed with British Sugar that these losses were direct, not consequential; NEI had to cover them.

The *British Sugar* case related to another area of technology, but the legal precedent applies to our industry as much as to any other (indeed it has already been applied in deciding a subsequent IT-related case). For an IT supplier, then, liability under contract will often be much larger than the supplier might suppose.

3.4.2 Goods v. services

Things traded normally come under the heading either of "goods" or of "services", and often the distinction is clear. A car is a "good", a driving lesson is a "service". Computer software seems to fall in between: should it count as goods or as services?

To an IT expert, the question may seem silly. Software is what it is; if it does not fit these categories clearly, too bad for the categories. But in law these categories are crucial, because the nature of a supplier's liability for defects depends on them. If you supply a service, the law requires only that you act with due care, not negligently. If you supply goods, you have an absolute obligation to supply goods which are reasonably fit for use; if they are not, it is no defence to say "That is not my fault, I had no way of knowing about the defect."

To understand the rationale of this longstanding distinction, think for instance of a doctor, who provides a clear example of a service (even if nowadays, under the NHS, most patients do not pay for it). We cannot demand specific results from a doctor, for instance we cannot insist that all his patients must be cured, though we do expect him to exercise the levels of skill and care that are normal within his profession, and if things go wrong through his negligence he may be sued. Contrast that with a greengrocer, who sells goods. If a greengrocer sells mushrooms which are poisonous, we do not want him to escape liability by saying "I didn't realize there was anything wrong with them." We need the greengrocer to ensure that he sources his mushrooms in a way which leaves him confident that they are safe, and if he is not prepared to do that then he is in the wrong job.

Is the software engineer more like a doctor, or more like a greengrocer? We might feel that a software engineer is much more like a doctor, in terms of the subtlety of the work and the impossibility of ensuring that outcomes are always perfect (and evidently, in terms of legal liability, it is preferable to be a provider of services rather than goods).

But one reason why society is willing to hold doctors only to the standards of care normal in their profession (rather than making absolute demands about outcomes) is that the medical profession defines and enforces high professional standards. Rules are laid down by the General Medical Council, and every now and then we read that some delinquent doctor has been struck off the register of those allowed to practise.

Is software engineering a "profession" in this sense? If so, how are its "normal levels of skill and care" defined and enforced? As we saw in chapter 1, we have a professional organization, the British Computer Society, which attempts to define standards of professional practice; but only some IT workers apply for its qualifications. The BCS maintains a register of Chartered Information Technology Professionals – but I have never heard of it striking anyone off its register, and if it did I am not sure that newspapers would bother to report it.

All this may change. Until it does, we perhaps cannot complain if the law classifies us with greengrocers rather than doctors, and accepts no excuses when software is unfit for purpose.

To date, the legal issue is open: it simply is not settled which side of the goods/service boundary software falls, despite the potential importance of the issue for suppliers. There are classic cases which illustrate how thin this boundary is. A dentist who makes a set of false teeth draws on a great deal of professional skill, and must tailor the work closely to the individual client's needs: but it is settled law that false teeth are goods, not a service. Conversely, when someone commissions a portrait from a painter what he gets is a purely physical object, a canvas covered with pigment: but portrait painting is treated by the law as a service rather than supply of goods.

With software, one can argue either way. When a client commissions a bespoke one-off software system to meet a specialized need, the client is paying for programmers' intellectual skill, not for the physical disc it is delivered on. On the other hand, a standard off-the-shelf software package might seem more like goods, even though (if it is delivered online) nothing physical changes hands. These are two ends of a spectrum with plenty of intermediate cases: for instance, a standard package may be adapted to a certain extent to suit an individual client, or a bespoke software system may be sold not separately but in a bundle with hardware (which is certainly "goods"). The question where software stands with respect to this legal distinction will remain open until future cases establish a body of concrete precedents.

3.4.3 Implied terms

Contracts aim to achieve precision by spelling details out explicitly, but no contract spells *everything* out. It is not possible: there is no limit to the range of considerations that could turn out to matter in some future dispute. One way in which the law addresses this problem is by, in effect, rewriting aspects of a contract which comes before its notice in a dispute. The law will add extra, "implied" terms to those which appear in black and white. (In the next section we shall see that the law may also cross out some of the terms which do appear in writing.)

One type of implied term relates to *business efficacy*: if a contract fails to make commercial sense without additional wording, the court will supply that wording.

An IT-related case was *Psychometric Services* v. *Merant* (2001). Merant contracted to produce software to enable Psychometric Services to run its business online, but the object code delivered by Merant proved not to work adequately. Psychometric Services asked the court to order Merant to hand over the source code, so that (having lost faith in Merant) it could get the system completed by someone else. The contract did not state that the client was entitled to the source code, and a supplier will commonly keep this to itself so as to ensure future business from the client. But the judge noted that the contract bound Merant to maintain its system for no more than two years; after that, if the client had no copy of the source, "none of the inevitable bugs [would] be able to be fixed. No development [would] be possible", and Psychometric Services would almost certainly go into liquidation. That would mean, the judge said, that "the agreement made no commercial sense at all"; so the contract was to be read as containing an implied clause giving Psychometric Services the right to the source code.

Another common reason for adding an implied term will be that the supplier knows how the client intends to use the goods. In that case, even if the contract does not make the intended use explicit, the supplier will be required to supply goods suitable for that purpose.

This is a sensible rule in principle, but in practice it can be hard to say what counts as suitable. A classic precedent was set long before the computer age in *Griffiths* v. *Peter Conway Ltd* (1939). Mrs Griffiths ordered a bespoke tweed coat from the tailors Peter Conway. When she got it, she complained that it brought out a rash on her skin, which was unusually sensitive. Her case was that Peter Conway knew the coat was for her to wear, and this coat was not suitable for her, so they were in breach. But the court decided that there was no breach, because although Peter Conway knew Mrs Griffiths intended to wear the coat herself, they had no way of knowing about her sensitive skin.

With software, problems like this occur in spades. Mrs Griffiths could have warned her tailors about her sensitivity, if she had thought to do so; but in IT, as Rowland and Macdonald put it (p. 138), "at the time when a contract is made, it may be difficult for the parties [either of them – GRS] to accurately define the software required".

By now, courts do understand that under software contracts one cannot require suppliers to get things right first time. That was established in *Saphena Computing Ltd* v. *Allied Collection Agencies Ltd* (1995). Saphena contracted to produce a system for a debt-collecting agency, but their system proved unsatisfactory; the two sides agreed to terminate the contract so that Allied could find an alternative supplier. Allied argued that the inadequacy of Saphena's system put it in breach of contract (so that Allied would be entitled to withhold payment). But in his decision the judge quoted with approval the evidence of an expert witness for Saphena: "no buyer should expect a supplier to get his programs right first time. He … needs feedback on whether he has been successful." Thus it seems that contracts for software will be interpreted as giving the supplier the right to test and modify its system over a reasonable period (which would not always be so for contracts in other business domains).[14]

That point might seem to suggest that a supplier of imperfect software is fairly safe. But the *St Albans* case to be discussed in the next section means that suppliers are not as safe as all that.

3.4.4 Unfair terms

The tradition in English law was almost total freedom of contract. By and large, two parties could agree whatever contractual terms they pleased, and the law would enforce them. This began to change towards the end of the nineteenth century, and the present situation is rather different. The law will refuse to enforce various explicit terms in a contract as "unfair". The statute currently applying is the *Unfair Contract Terms Act 1977*.[15]

Unfair terms fall into two classes. Some terms will be struck out in any circumstances: a clause excluding liability for death or personal injury will never be valid. More interesting for our purposes are cases where some term is regarded as unfair in the context of the particular contract in which it appears.

The motive behind the doctrine of "unfair terms" is society's wish to make bargaining power more equal as between the "little guy" and big business. However, the effects of the law extend more widely.

Take the case of *St Albans City and District Council* v. *ICL* (1996). ICL was then the leading UK computer supplier (it has since been taken over by Fujitsu), and it supplied St Albans with software to calculate the poll tax (the unpopular system of financing local government which operated for a few years before being replaced by the council tax that we know today). Poll tax was charged at a set rate per head, decided annually by each district council. A council knew what its total budget was, so it arrived at a figure for poll tax by dividing that total by the number of taxpaying residents. Unfortunately, ICL's software contained a bug which had the effect of overestimating the St Albans population, meaning obviously that the poll tax figure was set too low. The loss to the council was £1·3 million.

The contract limited ICL's liability for software faults to whichever was less of the price paid for the software, or £100,000; so St Albans would have been seriously out of pocket. But the court struck this limitation out as unfair, and ICL had to compensate the council fully. Grounds for the judgement of unfairness included the following (as well as some other points we shall not go through here):

- ICL was an organization with more resources than St Albans (which was true, though a city council in South-East England is not most people's idea of a "little guy");
- ICL had product liability insurance under which it could claim, whereas (according to the judge) one could not expect a local authority to insure against commercial risks (a number of commentators wondered "Why not?" – but the judge was the judge);
- St Albans had tried to renegotiate this particular clause, but being up against a tight deadline they did not succeed. By law, a council must send out its annual tax demands by a certain date, so St Albans had to have some system in place by then.

So, although the law recognises that bugs are unavoidable, if a bug has particularly expensive consequences an IT supplier cannot always rely on a cautiously-worded contract to protect it from those consequences. What counts as "unfair" has an unavoidable element of subjectivity. The trend of unfair-terms decisions related to IT has been so adverse to suppliers that by 2001 the profession was asking "Do the Courts have it in for the IT industry?" (Since that date, Jeremy Newton sees signs that the tide may have turned somewhat in favour of suppliers.[16])

3.4.5 Development risk

If courts have appeared unduly harsh towards software suppliers whose products are less than perfect, this may be partly because the law has not appreciated how much innovation and unpredictability is involved in our industry. Many IT professionals may feel that it would be quite unreasonable to treat an unsuccessful software system as proving that the developers of the system must have been culpably negligent: it is not like building a bridge, where the engineering issues have been settled for some time and perhaps a qualified bridge designer really does not have much excuse if his construction collapses. To quote Rowland and Macdonald (p. 235):

> An important consideration for a technologically advanced industry such as the software industry is the legitimate concern that innovation should not be stifled by legal rules. Designs for systems that are "at the cutting edge of technology" may not have been tried and tested in the same way as a more pedestrian project, and the industry owes its success to its ability to create and market new methods of control or new systems and products.

Perhaps the law ought to regard a measure of what is called *development risk* as inescapable.

However, computing is not the only industry which innovates, and the law has taken a hard line with other industries where innovation has proved dangerous. Rowland and Macdonald cite *Independent Broadcasting Authority* v. *EMI and BICC* (1980), a professional-negligence case which eventually reached the House of Lords, stemming from the collapse in 1969 of the television transmitter on Emley Moor near Huddersfield. In its day the Emley Moor mast was one of the tallest freestanding structures in the world, designed in an innovative way by BICC (now Balfour Beatty) and built by EMI (which used to be a manufacturing as well as a music company). It was brought down by a combination of ice and high wind. Defending themselves against the accusation of negligence, BICC

> argued vigorously that a finding of negligence would be likely to stifle innovation and inhibit technological progress. They produced evidence that there was neither any available source of empirical knowledge nor agreed practice; they were "both at and beyond the frontier of professional knowledge". (Rowland and Macdonald, *loc. cit*.)

The Lords did not accept this as an excuse, and found that BICC's design was negligent. Quoting the judgement:

> The project may be alluring. But the risks of injury to those engaged in it, or to others, or to both, may be so manifest and substantial, and their elimination may be so difficult to ensure with reasonable certainty that the only proper course is to abandon the project altogether…

By good luck, when the Emley Moor mast fell no-one was hurt – but they easily might have been. Thus the supreme court of the UK has laid down that where such risks exist, innovation is not a defence against the allegation of negligence: what a responsible professional is expected to do is to refrain from embarking on the project.

One way of looking at this is that development risk may be inescapable, but the law wants the risk to be borne by the people who practise the innovative technology, not by their clients or by third parties. IT practitioners ought to be in a better position than others to evaluate IT risks and decide whether they are too great to proceed.

Nowadays IT is being deployed in many safety-critical applications. So far there seems not to have been an IT case analogous to *IBA* v. *EMI and BICC*, but this is surely only a matter of time. Our profession may often be oblivious to the legal risks it is running in this area. If you design a transmitter mast, you cannot fail to be aware that you are dealing with a tall and heavy construction exposed to all weathers, whereas computer code tends to insulate those writing it from the physical realities it is destined to control.

3.5 Torts

Mention of safety-critical applications brings us to the issue of torts. Because no-one was hurt at Emley Moor, there were no tort cases; BICC, EMI, and the IBA were in contractual relationships with one another, whereas if a passer-by injured by the collapse had sued one or more of these parties the case would have come under the "tort" heading. IT is used routinely nowadays in applications such as fly-by-wire aircraft, or computer-controlled administration of drugs in hospitals. What would the legal situation be, if bugs in the relevant software caused an aeroplane to fall out of the sky, or a fatal overdose to be administered to a patient?

At the time of writing, there has been no new statute law relating specifically to IT-mediated torts, and, what is quite surprising, no significant cases have come before the courts yet. So anything said about how existing tort law will be extended to cases where IT is crucial can be only educated guesswork.

3.5.1 Strict liability for products

Consider for instance the *Consumer Protection Act 1987*, which implemented the requirements of the European *Product Liability Directive*. Before that Act, an individual who was harmed in some way by a product could take the retailer to court under the contractual relationship between them (whenever you buy so much as a bag of crisps, legally speaking you and the shop are creating and fulfilling a contract); but it was not easy for an individual to take legal action against the manufacturer, since there was no contractual relationship between manufacturer and consumer and to establish a tort it would have been necessary to prove negligence by the manufacturer. Yet the manufacturer might often seem the appropriate target for litigation. If its products are harmful, it is the manufacturer rather than retailer which is in a position to cure the defect or withdraw the line from the market; and, if the harm is serious and calls for a serious level of compensation, the manufacturer may have deeper pockets than a corner shop.

The Consumer Protection Act has the effect of imposing "strict liability" on the producer of a product. No longer is it relevant whether the producer acted in a blameworthy way; to render the producer liable, one need only establish a causal link between a defect in the product and the damage arising.

For our profession, the question then arises whether a software system is a "product"; legal experts have discussed this at length. It sounds like the same question as whether software is goods or a service, but "goods v. service" is a distinction rooted in English law. Because the Consumer Protection Act implements a European directive, it has to use the separate European legal concept of "product". As things stand, we do not know for sure whether software counts as goods, and we do not know whether it counts as products either, but when relevant decisions arise it could turn out that the answers to the two questions will be different.

Jane Stapleton suggests that the European legal system is likely to interpret "product" widely, to include software even if English law classifies it as services rather than goods, because the fallible human activity which is the hallmark of services is "masked" in the case of software. When a customer visits a hair salon she physically witnesses the stylist exerting professional skill, whereas it is hard to see past pages of program code to the programmer toiling in his cubicle.[17] Evidently, for the welfare of our profession we should hope that software does *not* count as a European "product", but the question is impossible to decide *a priori*: we must wait to see which way courts go.

If software is counted as a product so that the Consumer Protection Act creates strict liability for damage caused by bugs, there will be a further issue which is perhaps more problematic for IT than analogous issues would be in other domains. What counts as a "causal link" between a software bug and damage arising in connexion with it?

Rowland and Macdonald (pp. 222–3) refer to the notorious 1980s episode in Canada and the USA when faulty software led the computer-controlled Therac-25 radiotherapy machine to administer excessive doses of radiation to a number of cancer patients, killing some of them.[18] In this case the causal link is clear, but what (Rowland and Macdonald ask) if the bugs had happened to work the other way, so that patients received too little radiation? Then, some of the patients would have died from cancers that could have been cured. Legally speaking, would there be a "causal link" between the software defects and the deaths – or only between the cancers and the deaths?

3.5.2 "Development risks" in the case of torts

In one respect, our Consumer Protection Act explicitly differs from the corresponding laws in some other EU countries, although all were introduced to implement the same Directive. The European Directive gave EU member states a choice over whether or not to include a "development risks" defence in their implementing legislation: if a product turns out to be harmful, is the producer allowed to escape liability by arguing "that the state of scientific and technical knowledge at the time when he put the product into circulation was not such as to enable the existence of the defect to be discovered"?[19] On the one hand, not allowing that defence "might discourage scientific research and the marketing of new products". On the other hand, allowing it might leave the new legislation fairly empty.

Some EU countries did not include a development risks defence in their implementation of the Directive. The UK did include it, and in fact the form of words in our Consumer Protection Act is so broad that the European Commission took proceedings against the UK for failing to implement the Directive properly. (However, those proceedings failed, and the Consumer Protection Act stands.)

This might suggest that British law will be reasonably merciful to producers of software which does unforeseen harm (even if software is counted as "products"). But development risk is about things that in some sense push the boundaries of current human knowledge. Very often, when software bugs cause harm, this will not be because of limits to our scientific knowledge about the consequences of any specific bug, but merely because it is so difficult to locate and eliminate every last bug in a complex program. Each individual bug may be recognisable as an error once it is found, but no matter what régime of testing is applied before the package is released, some bugs are missed. How much testing does it take to discharge one's legal responsibilities?

We saw, above, that English contract law accepts that some bugs are inevitable. But we are discussing tort law now, where harm is done not to trading partners but to third parties; and in this area, while there are no IT-related precedents as yet, what precedents do exist suggest a much tougher line.

A frequently cited case is *Smedleys Ltd* v. *Breed* (1974). This was not in fact a tort action but an appeal against a criminal prosecution under the *Food and Drugs Act 1955*; and that Act has been superseded by newer legislation. But neither of these points are seen by commentators as necessarily important; the case set a standard for the required level of quality control with respect to risks to third parties.

Mrs Voss bought a tin of Smedleys' peas at Tesco's, and opening it she found a green caterpillar among the peas. The resulting case went as far as the House of Lords, which accepted that Smedleys carried out extremely thorough mechanical and manual testing to guard against foreign bodies in its food production; statistically speaking they achieved an impressively tiny incidence of complaints. (In the judgement, Lord Hailsham also pointed out that even if Mrs Voss had not spotted the caterpillar, being thoroughly cooked it would have done her no harm – she "could have consumed the caterpillar without injury to herself, and even, perhaps, with benefit.") But none of this got Smedleys off the hook. The conviction they were appealing against was upheld, because if they had examined that caterpillar during the testing process they could have recognised it.

In other words, *no* amount of testing is sufficient, if it leaves some individual defects which could be recognised as defects in the current state of human knowledge. It is irrelevant that the overall incidence of defects may be as low as current technology permits.

The analogy with software testing is uncomfortably close. Even if it is accepted that the "last bug" in a program can never be found, that fact looks unlikely to help a software developer whose undetected bug leads to a tort action. Indeed, Lloyd (p. 569) argues that the law will see the software developer's liability as specially clear. A caterpillar is a natural object, but "With software, the producer is put in the position of creator…the producer cannot disclaim knowledge of his or her creature's properties." So, at least, the law may assume.