14

Treat people like computers?[†]

Designing usable systems for special people

Harold Thimbleby

Editor's introduction

It was pointed out in the Preface that the term handicap applies to the effects a person's limitations have on their ability to function in their environment. This chapter explores that concept in a very important direction: it advances the proposition that the average human-computer interface is so badly designed that *anyone* who uses it is handicapped by it. Moreover, it suggests a practical way to start to respect the user, based in the theory of computer science. It should therefore appeal to the computer scientists and programmers who design interactive systems.

14.1. Introduction

Three true stories:

- An old and disabled woman makes a special trip to visit her electricity utility, to make a payment on her monthly credit book. The clerk refuses to accept payment because the date for the next instalment is tomorrow, and the computer cannot process early payment. The woman goes to the local newspaper, and the story that gets printed includes the clerk's manager's complaints about the computer system: it is not only badly designed for customers, but it also disables staff in how they can handle customers.
- I was recording a live TV broadcast on my video recorder. The adverts came on, so I pressed Pause, to pause the recording. When the adverts finished, I pressed the wrong button, and the VCR did something I'd never seen before. I got confused. The frustration I felt was so intense, I did not want to waste my time with this gadget ever again! — effectively disabling myself from gaining its possible benefits in the future.
- The ticket regulations for travelling with British Rail are so complex that passengers can often be seen anxiously trying to understand their ticket, as the Senior Ticket Collector for the train announces that *only* he is sufficiently qualified to answer any queries. The ticket regulations are so complex that they disable people.

Thus badly designed systems handicap all users. Attitudes, too handicap users: users are generally blamed for their own difficulties. *Didn't they know*? *Didn't they read the manual*? *Are they too old for this technology*? The designers thereby avoid any responsibility for the users' difficulties. Their attitudes handicap users and deflect attention from the engineering design of systems.

[†] To appear in *Extraordinary People and Human-Computer Interaction*, A. Edwards, editor, Cambridge University Press, 1994.

In this chapter, it will be argued that improvements should be sought in better engineering, and that there are the means to do so. This chapter may make the reader question who is handicapped and what is the cause of that handicap.

Inevitably, this chapter complains about the disabling design of interactive systems, although a longer chapter would also include disabling designs in intelligent homes for the elderly, through to all areas of interactive technology. If the reader thinks that such complaints are overdone, or that the problems complained of are obvious — then why are such problems so common? Why do designers or manufacturers not notice them and not avoid them? *They cannot be so obvious to designers, or perhaps designers, despite knowing, do not know how to recognise or avoid the problems.*

As well as providing ammunition for people who want to complain about the problems, more constructively this chapter will make two central contributions. First, we will point out an analogy with automobile safety: informed consumer pressure can change things, and usability (drivability and safety) *is* an engineering issue. Second, we will point out an analogy between user and computer: designers (who can program) implicitly know how to do the engineering properly. We discuss the imbalance that computers get better treatment than users, to the point of making the almost outrageous suggestion that if humans were treated with at least as much care as computers are, then computer users' lives would be improved.

14.2. The automobile analogy

In 1963 the American Association of Casualty and Surety Companies reported a common fault found in many cars, that they sometimes ran back against the parking brake. The solution they suggested was to educate the driver to get into the habit of putting his foot on the foot brake pedal when pulling on the parking brake. The foot brake would apply a greater force on the brake shoes than could be applied by hand alone, thus ensuring that the shoes engaged properly and the parking brake held. The advice they provided was that there was no problem … if a driver trains himself to do this. Put like this, the solution clearly lay in the area of driver psychology: most drivers, for instance, did not read the car manual. How could drivers be educated to drive properly?

This problem was reported in Ralph Nader's now classic book *Unsafe At Any Speed* (Nader, 1965). But looking back, with the benefit of the consciousness raised by Nader's book it is now obvious that a proper solution would have been to solve the engineering problems in the parking brake. The parking brake should have worked properly, and not have required the additional assistance of the foot brake! The solution was not in getting the driver to read the manual, but in designing the system such that no comment even needed putting in a manual. This point of view is now widely accepted. Yet in computer systems, we can see from the standard warranties provided with proprietary software that designers or manufacturers are not and do not *want* to be responsible (Thimbleby, 1990a and 1990b). The purchaser is given no effective come-back on product defects, of which there are many. There is a veil of secrecy over crippling product defects.

Today, computers, whether PCs or embedded inside domestic products and other machines, are designed quite as badly as 1960s cars. Most are partially unusable. They rely for their apparent success on the long sufferance and dogged perseverance of the user. The similarity between the poor design of cars thirty years ago and modern computing systems is indicated by the agony columns in many computing magazines, and the sorts of solutions to readers' problems that are suggested there. Hence the title of an article, "You're right about the cure, don't do that" (Thimbleby, 1990a), reflecting on a piece of agony column wisdom, where a reader was told that the only way around a bug was to give up and avoid a useful facility.

Computers are (occasionally!) useful, and users are very often justifiably pleased with what they can achieve with their help. Indeed, the more so the less the user could do, previously, in comparison with what they are now able to do with computer support. That may explain why users tend to put up with computer limitations without complaining: on balance, users are better off with computers than without them. When a computer crashes (due to a software error) the user, like a laboratory rat, soon learns not to do whatever caused the crash — that is, if the user is fortunate enough to see any discernible reason why the crash occurred. Soon, the user will learn how to work the computer reliably, or modify their work so that only reliable parts of the computer system are required.

Another reason why users tend to down-rate usability problems is that the computer represents a major investment in financial terms and training time. Admitting problems is tantamount to admitting wasting time and money. It is easier to believe that the computer is 'alright,' and that with even more effort, more money or more training, one would get used to it. With today's consciousness, it seems easier to admit being technically incompetent than to wasting time and money.

This is the same user oppression as the 1960s driver oppression: engineering problems are passed off as user handicaps, and the users acquiesce.

If problems are the user's handicaps, who is responsible for the problems? Not the designers, manufacturers or dealers! Should we have a government initiative in technology literacy, so that users can better put up with the oppression? Should users (perversely) feel proud with the sophisticated skills they learn (Thimbleby, 1993)?

It is a truism that we, as humans, are all handicapped with respect to our potential. The discussion above indicates that everyone is reduced by current technology. If you start off being handicapped, you are made much worse off, in ways that are not always obvious, but in ways that both manufacturer and user conspire to conceal and deny.

The exploitation of users is technically avoidable. Although improvement will result from action at all levels, political, consumer action, litigation, human factors consciousness, the aim here is to argue that technology *itself* has better to offer users. Improvement will result from better engineering, and a more professional attitude from designers.

14.3. The computer analogy

A user has to put up with poor systems. A user has to adapt their work patterns to fit in with or avoid quirks and bugs. Yet if the human user was instead a computer, and had any such problems, then obviously the designers would have to fix the problem. A computer cannot choose to do other than what it is told, and therefore it is obvious that its problems (say, dividing by zero) must be *caused* by bad programming. It is not sensible to criticise or blame a computer for being uninformed, stupid, old or incapable. Its problems *have* to be fixed by the programmer responsible. When a user has problems, however, responsibility is evidently not so obvious.

In all of the many millenia before the age of computers, people were talking to each other without worrying too much about their own idiosyncrasies, their grammatical errors, their interruptions, their ways of resolving misunderstandings, and so forth. All humans share the same skills and the same defects, and so they are hard to see. We (as humans) often do not realise just how limited and error-prone we really are.

Today, many people interact with computers, but computers have *very* different skills and defects than humans. Thus when ordinary people interact with computers, they become acutely aware of the mismatch between their skills and the computers'. The very human skills of conversation repair do not help with computer defects any more than a computer's skill helps it cope with human quirks. The computer makes no concessions for the differences and incompatibilities: always, it seems it is the human who has to adapt to the idiosyncrasies of the computer.

We must ask: “What is common to both computers and humans, and how can we ensure that interactive computer systems are designed to be at least up to the common baseline?”

At least the first part of question is a familiar and foundational issue in computer science. There is a standard and uncontentious answer, and it provides a rigourous standard for what is minimally acceptable behaviour in programmable systems — whether human or computer. We will briefly discuss the standard answer, and then we will be able to argue that this standard is *higher* than present computer systems manage to achieve; yet because the standard is so straight-forward and well-researched, there are very many practical opportunities to improve the state of the art. Otherwise, until systems are routinely designed to this standard, humans will be brought down to the subhuman standards of badly programmed computers.

Turing Machines were defined (by Alan Turing) specifically to get rid of the distraction of special and particular skills to determine exactly what, in a formal sense, a person could do. People are Turing Machines.

If we say that humans are Turing Machines, we run the risk of being misunderstood. Turing Machines sound like robots, and people are certainly not robots. However, an anatomist would have little problem understanding a human as a particular sort of mechanical machine, whose levers were bones. A physicist, going to a further extreme, would have little problem understanding a human as a point whose only characteristic was mass. Neither view demeans the human. Likewise a computer scientist should have little problem understanding a human as a Turing Machine. Saying this is no more contentious than saying “drivers are point masses” to a physicist. People obey the laws of mechanics in quantifiable ways; ergonomics is the discipline that uses their mechanical properties to make people’s environment and work more comfortable and effective for them.

The Church-Turing Thesis posits that all forms of computation are fundamentally equivalent. There is considerable weight of evidence for the Church-Turing Thesis, and it has the standing of a natural law. Perhaps there may be an experiment (as yet to be devised) that refutes it, but until then we may believe that everything, whether computer, human or otherwise, acts in accordance with the laws of computation.[*] The important point is that what is impossible for a Turing Machine to do is impossible for any other concept of computing, *including* human beings, be they disabled or exceptionable. Such problems are termed non-computable, in distinction to the computable problems that computers can solve.

When a car designer makes a car that does not protect the driver from minor collisions, he is implicitly hoping the driver can miraculously escape the consequences of physical laws. The unfortunate driver who obeys the law of conservation of momentum — and he cannot fail to do otherwise — has an unfortunate tendency to fly through the windscreen or to hit the steering wheel. (One wonders why safety belts took so long to be adopted.)

Likewise, by accepting the Church-Turing Thesis, we are able to make a distinction between two sorts of usability problem. There are designs that merely *hinder* the user, and there are designs that try to make the user *break* the natural law by requiring them to do something non-computable. It happens rather more often than

[*] There is a technical quibble about quantum mechanics. But nobody is suggesting that to rely on quantum mechanical effects at the human level would be advisable. The same goes for intuition and spiritual aspects: these are subtle human qualities that computers apparently do not share, but to rely on them would generally be a burden to impose on users.

might be expected. (Perpetual motion machines are continually being invented, and some of them run for an impressive time before succumbing to reality!)

The consequences of setting humans non-computable tasks is as disabling as ignoring the conservation of momentum in the design of car safety features. There is no argument: designers must give users computable systems. It follows that users must have or be given an algorithm (i.e., a ‘program that does something computable’) to operate the system; the algorithm may of course be constituted in a vague way from the user’s world knowledge, or it can be more precisely defined in a manual or operator’s guide.

14.3.1. Styles of computation

The Turing Machine is only one of many equivalent ways of formulating computability. Strictly, we should have been writing ‘Turing Machine Equivalent’ above, since we are not trying to impose the Turing Machine’s *particular* approach to computation as an approximation of human computation. Indeed, since Turing Machines are formal conceptualisations, they conjure up a rather idiosyncratic approach to computation that would never be used for practical computation! The point, which is easily made, is that in principle *all* models of computation, including human, can be reduced to Turing Machine configurations. Just as mechanics can be reduced to a few basic types of lever does not mean that every machine is a heap of levers.

The Turing Machine approach is imperative, to some extent a Pascal-style of computing. There are several alternative, non-imperative approaches, represented, for instance, by the computer programming languages Prolog (declarative), ML (functional), and others. Just as a programmer chooses an appropriate computational model from this wide choice, we intend an interactive systems designer to make a similar wide choice about the style of interaction based on the user and the user’s specific tasks.

For a worked example see Thimbleby (1990b, chapter 15), which is a discussion of logic programming applied to user interface design. For exactly the same reasons that logic programming is promoted as an approach to programming, for user interfaces it provides additional flexibility.

14.4. What impact for design?

A precise and brief summary of the above discussion is that systems that are meant to be used by people should support computable algorithms for their use. (For some tasks, like education, one might want mystery and surprise, but this should be done deliberately rather than by default whatever the user is supposed to be doing.) Without algorithms, systems are far harder or more error-prone than they need be.

We give some examples now of the change in perspective this view brings to user interface design. None of the examples is pursued in much depth; nor is the repertoire by any means exhausted by this short list. The purpose of the examples is to show the potential of the approach, and secondly to stimulate the reader.

14.4.1. Physical disability

Various forms of physical disability reduce the speed and energy that can be put into conversation and typing. There are many algorithms for text compression: if typing is a burden for the user there are algorithmic approaches to improve the user interface. It is pleasing, then, to report on the success of some recent work motivated in this way (Darragh and Witten, 1992, and see also Chapter 6), that is an application of computing theory to usability.

14.4.2. A word processor

Few word processors distinguish in the way they display spaces, tabs and nothing at all (e.g.,the nothing beyond the end of a line). All are blank regions on the screen, yet they behave differently. Suppose now that a user wishes to move the word processor's cursor from where it is, to land on a position they have in mind, perhaps to correct a spelling. Because blanks can be different things, the user is in principle unable to work out (that is, is unable to *compute*) how to move the cursor to the desired goal. Word processors that use a mouse have a similar problem, and one that is sometimes more mysterious. There are places on the screen that the mouse can be positioned in, yet where it is not possible to type. How does a user detect such places without wasting time?

This is a very simple, but common, example of a user interface assuming that the user can somehow circumvent the limitations of computability. In practice, therefore, the user must experiment: try and move the cursor where it is wanted, and if it ends up somewhere else, try to move it from there to where it should be. As a rule, the user will be able to do this, so the way the cursor moves will not be that surprising. Nevertheless, the word processor requires the user to solve a problem that a computer would fail at. (It would be amusing to challenge a programmer to write a program to use a word processor via the standard user interface of that program.)

Software designers get away with nonsense (like the example illustrates) and everyone else gets so used to it that they even start to expect it. Indeed, many expert word processor users may fail to see the problem as described so briefly here. Users, and some designers, evidently, are being trained to be masochists. It is an impressive brain-washing manoeuvre that makes handicapped users feel proud of skills that keep them handicapped.

The proper solution to the word processor problem is trivial: design the system so that the cursor motion is not affected by what the cursor is moving over (then the *up* arrow key, for instance, will *always* move directly *up,* which is reasonable). There are many other solutions (Bornat and Thimbleby, 1989). The point is: users put up with a little unnecessary mental effort, some surprises, and some unnecessary but unavoidable experimentation, in this example, to get a cursor exactly where they want it. A computer would not readily tolerate such unpredictability: why should a human?

A modern programmer, writing a program, not for a human but for a computer to do the editing, would say that tabs or spaces is a detail of implementation and not the concern of high level operations. Abstract data types and encapsulation are standard tools for programming computers; why are they not standard tools for user interface design? Why do the underlying ASCII code details intrude into the user's editing task if it would be avoided for the computer's sake?

14.4.3. Dementia

When a person with dementia is hospitalised or moved to a new home, they can become disoriented and may deteriorate seriously. Part of the reason for this is the lack of familiar cues in their surroundings: they are literally lost.

Computers don't so much get lost: they never knew where they were to start with. Thus any algorithm for a computer to move around in a conceptual space (a data structure) must provide for keeping track of what places have been visited, and what places looked promising for future exploration. There are very many algorithms for doing this systematically, two of the simplest being breadth-first search and depth-first search (Thimbleby, 1991b).

No current hypertext systems provide algorithms for their users to avoid the problems of them getting lost. Instead, users have to rely on their memory, and they become disoriented, a phenomenon called 'getting lost in hyperspace.'

If this is what happens to people with dementia, they might be helped in a similar way, by providing the cues that a computer would need to do their tasks. The advantage of the computer analogy is that it makes abundantly clear what cues are necessary, though in practice more alternatives might be needed.

In an institution with lots of confusing passages, radio devices could be used, not only to say where one was, but also to say where one had been — just as one would provide flags (or whatever) to stop a computer from getting 'stuck in a loop.' In fact, demented people live in the past and would probably not be able to benefit from such novel technology. But maybe it should not be novel; maybe it is time that as direction aids (for car navigation) become more widely available they should be better designed, along these principles. This would improve them, and would help their users as they age and start to rely on their facilities for day-to-day living.

14.4.4. Permissiveness

Most people are right handed. Although left-handed people grow accustomed to an awkward right-handed world, a person with a stroke can suddenly lose the use of their right hand and find things very difficult. They are additionally disabled by any device, such as a door knob, that assumes the correct way to be operated is to be manipulated in a right-handed fashion. More generally, the underlying assumption is that there is *one* correct way of doing things, in this example, the right-handed way. Users therefore have to conform to 'the correct way' chosen by the designer.

Suppose we are designing a network program, or an algorithm for computers to talk to each other. We cannot assume that another computer will work the way we deem to be correct. Any compatible program must cater for several ideas of 'correct' protocols. The successful network program has to be designed so that it can cater for various protocols. If we called the protocols left- and right-handed, the point would be clear: computers never learn or adapt (change handedness), so we have to cater for both.

The assumption that there is just one correct way to design any human facility, even for disabled people, is so deeply entrenched that it is useful to have a word for deliberately avoiding the narrowness. A system is *permissive* if it permits itself to be successfully used in more than one way.

The video recorder example we started this chapter with required the user to press the correct button. Different manufacturers make different design decisions about which button is correct. In a permissive video recorder, any of the other buttons that don't do anything at this point could be used. Then whatever a user does would be correct, whether they press the Play, Record or Pause buttons.

The user of a permissive system need not even know that there are alternatives. The old lady (of the chapter's first example) faced a non-permissive credit control system; railway travellers (the third example) face a non-permissive, and obscure, set of ticket regulations.

(Even human factors experts may assume there is *one* right design and users must know it. A case in point (in Nielsen, 1993) was a permissive system that provided users with alternatives, yet the system design was criticised on the basis that users did not know how to use it properly if they knew only one of the alternatives!)

14.4.5. Know the user

Few programmers would try to make a program they called efficient without knowing about the computer it was running on. Few competent programmers would try to optimise a program without first making measurements about its performance.

Why then do programmers make systems they call usable *without* knowing about the users who it works with?

Viewing user interfaces as the realization of algorithms naturally suggests exploring the many standard books on algorithms (such as Sedgewick, 1988) for ideas for developing quality user interfaces. This is the first step in making user interfaces more usable. The second step is to find out about the specific users.

14.4.6. The myth that children can use things adults can't

Finally, it is claimed that children can cope with modern technology better than adults. Children can programme video recorders when their parents can't. This point of view is disabling, for it suggests that adults are incompetent.

However, there is very little evidence in support of this. When one considers that children have not paid for the video recorder, are not worried about breaking it, that they generally have more time on their hands (they don't have a mortgage at the back of their minds), then their putative skills with video recorders look less dependent on age. Maybe the rest of us are not too old! Adults might be too busy, but few people are oppressed by being made to feel too busy rather than too old.

There is another possibility that I want to suggest, which is amenable to serious study. That is, children handle technology better than adults because they *don't* understand it. Adults have problems because they understand things 'too well,' and the technology fails to live up to their (quite reasonable) expectations.

In one simple, real, experience, my colleague Ian Witten and I expected a digital clock to work to the 24 hour clock: it certainly gave clues that it was a 24-hour clock, like being able to count from 00:00 to 24:59, even to 99:99. We were wrong (actually, the *designers* were wrong) and stuck, but the clock didn't tell us.

Experiments showed that if one made no assumptions about how the clock worked, if buttons were pressed at random, then the clock could be got working in just 9 or 10 button presses on average (Thimbleby and Witten, 1993). This is much quicker than we were: we took almost an hour. Since we had 'misunderstood' the clock, we would have been better off flipping a coin to decide what to do, rather than thinking carefully about it. Tossing a coin, acting randomly, might have led us to do the wrong thing some of the time, but we wouldn't consistently have done the wrong thing all the time, and be stuck — it might have been — forever.

It therefore seems plausible that children are successful with technology not because they are 'cleverer,' but because they do not understand what they are doing, and nor do they persist in doing any one thing. They play randomly, and that is a good way of learning how the world works, including digital clocks and video recorders. Adults, having spent most of their childhood playing, may prefer to work rather than play: certainly errors with a cooker clock or video recorder are rarely 'playful.'

It is in the manufacturers' interests to encourage us all to believe that we are inadequate when we cannot use their products. I was told I was too old when I complained that I could not use my video recorder. If I believe that, then my problems are my own fault, and I am handicapped.

14.5. Why engineering first? Why not human factors first?

When the functions of a product have been decided, perhaps a prototype built, it is common for designers (stylists) to design a suitable appearance. Users will experience certain sorts of problems that would be ameliorated by the provision or removal of certain features. However given the pressures of manufacturing in a competitive marketplace, by the time anything is known, the engineering effort will have been completed, and the product has to be ready to ship. Just when users can contribute to product evaluation, it is already too late to rectify design flaws. The solution can only be to *add* another feature.

When the same thing happened with cars, it was termed "the gadget approach to safety" by Ralph Nader. It was easier for car manufacturers to add gadgets (or let component manufacturers supply them to an after-market) than to make a well-designed car. There was an after-market in anti-sway bars for General Motors Corvair that greatly improved the handling of the car. General Motors should have made the car stable in the first place, or at least taken responsibility for fixing the instability problem they had created. Once a car is built, however, gadgets are all that can be provided. Likewise, there is a gadget approach to usability, whereby the more features a system has, the easier it is assumed to be. Whatever the user wants there is a feature available to do it. The marketing term for this is 'feature bouquet.' The bouquet approach not only misses opportunities for systematic design and rationalization, that might even reduce the number of features whilst increasing usability, but also encourages the provision of third party products to correct defects in the original product. When manufacturers rely on this approach they soon have less incentive to design properly in the first place.

Specific examples of how an engineering approach would impact user interface design have been developed and discussed elsewhere (Thimbleby and Witten, 1993).

14.6. Conclusions

Systems can be made easier to use, and there are rigourous technical standards available. A start in the right direction is to treat the user at least as well as a computer would be treated: to make the user interface computable, and to provide algorithms for the user to succeed in their work with the system. Present systems do less than this, and thereby exploit the user; and there is a culture to keep things this way. Sadly, users are caught up in this culture and blame themselves for not being clever enough, not being young enough, and not having read and understood the manual, they are not even qualified to comment.

In reality, rather too many usability problems are straight forward engineering problems. Car parking brakes should be designed properly. It is obviously not the drivers' responsibility to learn work-arounds or to read agony columns about driving techniques to avoid dangerous features. Yet when it comes to working with computers, or video recorders, or microwave cookers, the user lives in a world of restrictions and enforced work-arounds. When a user becomes pleased with their technological prowess manufacturers have profoundly handicapped somebody.

How can we encourage designers to respect users a little more? Even to respect them at least as well as they would computers? (The current standards of usability are substandard from a computer's point of view.) When we take the Church-Turing Thesis seriously we see that by treating users as non-computers, we are treating them as far *less* than human. In other words,

To treat humans as less than computers is disabling, and it is avoidable.

Humans are Turing Machines (and more besides) and are only held back by poor interfaces to bad technology.

Designers should work out algorithms to ensure that their designs are usable, at least in the computability sense. Only then should they go to human factors consultants to finish the design. Otherwise the necessary human factors contribution will only palliate or camouflage engineering defects, or even condone work-arounds! Too often cosmetics (including gratuitous features) are used to gloss fundamentally unusable designs.

In aircraft cockpits, we may talk of computer-encouraged error (Race, 1990), but the pilot is still blamed. In cars, radios have caused accidents, and this is hardly surprising when you look at one and try and understand it. Many have over twenty buttons with meanings that depend on what text 2mm high says. What driver can read that when concentrating on driving? Not to put too fine a point on it, accident survivors are often disabled for life. Such observations are poignant when seen in the context of Ralph Nader's efforts of thirty years ago to improve car safety. The mechanical engineering has improved, but the interactive usability has decreased.

References

R. Bornat and H. Thimbleby (1989). "The Life and Times of Ded, Display Editor," Cognitive *Ergonomics and Human Computer Interaction*, 225–255, J. B. Long and A. Whitefield, editors, Cambridge University Press.

Darragh, J. and Witten, I. H. (1992). *The Reactive Keyboard*, Cambridge University Press.

Nader, R. (1965). *Unsafe at Any Speed*, Pocket Books.

Nielsen, J. (1993). *Usability Engineering*, Academic Press.

Race, J. (1990). "Computer-Encouraged Error," *Computer Bulletin*, Series IV, **2**(6): 13–15.

Sedgewick, R. (1988). *Algorithms*, Addison Wesley.

Thimbleby, H. (1990a). "You're Right About the Cure: Don't Do That," *Interacting with Computers*, **2**(1): 8–25.

Thimbleby, H. (1990b). *User Interface Design*, Addison Wesley.

Thimbleby, H. (1991a). "The Undomesticated Video Recorder," *Image Technology, Journal of the British Kinematograph, Sound and Television Society*, **73**(6): 214–216.

Thimbleby, H. (1991b). "Can Humans Think?" *Ergonomics*, **34** (10): 1269–1287.

Thimbleby, H. (1993). "Computer Literacy and Usability Standards?" *User Needs in Information Technology Standards*, 223–230, C. D. Evans, B. L. Meek and R. S. Walker, editors, Butterworth-Heinemann.

Thimbleby, H. and Witten, I. H. (1993). "User Modelling as Machine Identification: New Design Methods for HCI," *Advances in Human Computer Interaction*, **IV**, 58–86, D. Hix and H. R. Hartson, editors, Ablex.