The Computer Science of Everyday Things

Harold Thimbleby

Computing Science
Middlesex University
Bounds Green Road, London, N11 2NQ

http://www.cs.mdx.ac.uk/harold

Abstract

Technology is fashionable, wonderful and getting better; Moore's Law predicts substantial, sustained improvement. Yet the usability of 'everyday things' is low (video recorders being a notorious example). It seems to follow that improvements must be sought in areas outside technology, such as human factors. But a premise is wrong: in fact, the technology — the embedded computer science — is appalling!

Obsolescence, a symptom of Moore's Law, hides flawed design: poor products are replaced rather than fixed. The poor quality of the computer science of everyday things is eclipsed by the hope for fixing today's problems with tomorrow's consumption.

This paper reviews Moore's Law and the usability of everyday things; it shows that professional computer science can improve usability with ease. Improvement will be essential when ethical and environmental issues become, as they will, unavoidable design criteria.

Keywords Moore's Law, User interfaces, Mobile phones, Programming user interfaces, Environment.

1 Introduction

Almost all modern everyday things — from mobile phones, cameras, central heating controllers, calculators, to wristwatches and airplanes — work because of embedded computer programs. While most are technically impressive, their usability is often low. Usability problems are not just irritating, but are costly and stressful, and hazardous in many contexts: whether the users concerned are airline pilots, office workers using photocopiers, or home users struggling with video recorders.

Proceedings of the Australasian User Interface Conference, Bond University, Gold Coast, Australia, January 29–February 1, 2001.

It is popular to argue that everyday things are harder to use than they need be, and that improvements must be sought in human factors, that is in the study of users and their tasks [16], and in the study of ecologies and context of use [15], etc. The lack of "user centred design" is the classic reason for the failure of almost all programmed systems [10].

The computer science of everyday things has ignored usability (we give examples below), and its standards are woeful. This factor of low usability is ignored not just by designers and usability workers but also by computer scientists. While bad computer science masquerades as good, even desirable, improvements to usability will be sought in other fields. This fuels the impression that computer science has nothing constructive to do with usability. Design thus becomes driven purely by human factors concerns, which in turn de-emphasises and disengages the professional application of computer science: it becomes mere coding, which causes more problems, thus reinforcing the assumptions.

Indeed, as industry can evidently sell to consumers without improving usability, business practice itself creates a source of usability problems that is again of non-technical origin.

Usability, then, as a field has become dominated by marketing, social, empirical and statistical methods alien to computer scientists. The broad emphasis on human factors encourages computer scientists, further, to dismiss usability as irrelevant to their focus. Indeed, in simple terms computer science appears remarkably successful (cf. Moore's Law, §2, below): superficially confirming that usability problems must be solved elsewhere.

But successful interaction clearly requires understanding of *both* humans *and* computers. User interface design is very much a proper part, and certainly not a trivial part, of computer science, as well as of the human sciences. There is a serious educational problem, both at the level of educating technologists to design and program user interfaces better, and at the level of encouraging consumers to demand higher standards of lifetime usability, which could easily be provided.

2 The "impressive" Moore's Law

Technology is getting better, and has increasing performance over time. We can represent this by a graph (see Figure 1) using a curve of positive slope (it is not necessary to worry about the precise shape of the line, or exactly what 'performance' is measuring). For any particular task the user has, some minimal level of performance p will be required. From the graph it is clear that there is a crossover when the lines intersect, at performance $= p$ and time $= t$. Before time t, technology is delivering inadequate performance; after t, technology can deliver more than adequate performance.

Norman [17], crediting Christensen [3], makes the following argument. Before the crossover time t, manufacturers need only promise technical capability (which is easy, since technology is getting better all the time). After time t, products get distinguished not by their performance, which is more than adequate, but by more subtle — and harder to supply — properties like usability. For technologies like wrist watches, we are long past the threshold, and they are now fashion items, that is chosen on criteria mainly other than technological. But for many everyday things, like word processors, we should also be well beyond the crossover. So why aren't word processors much better?

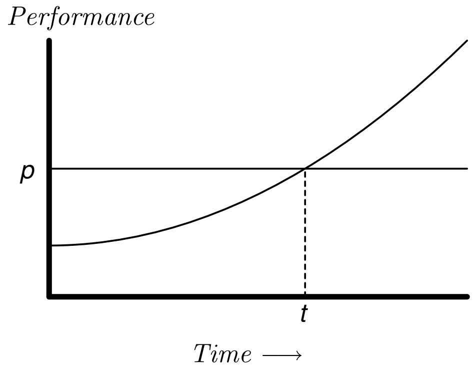

Figure 1: Perfomance of technology increases with time (curved graph). After time t perfomance exceeds the required level p.

Manufacturers can increase consumers' expectation of p, by providing or promising new 'essential' features: it is in manufacturers' interest to increase p, because this postpones the crossover. Thus many everyday things have numerous features, a factor influencing their usability.

The shape of the graph obviously reflects Moore's Law. Moore's Law, eponymous of Intel's Gordon Moore [13], says that computer technical infrastructure improves by a factor of two every 18 months. By infrastructure, take your pick: network bandwidth, processor speed, storage capacity, ... progress is relentless. We often do not wonder about Moore's Law beyond the excitement: everyone hopes for better products tomorrow.

The law has been followed closely and by many companies. If Moore's Law was an observation about scientific progress, we would expect more variance. We would not expect IBM, Intel and Motorola to be neck and neck; we would not expect the law to apply to the USA and to Japan. It is too coincidental [2]. More likely, then, the law is an observation about capitalism rather than a surprisingly uniform rate of technological breakthrough.

Stimulating consumer demand by enhancing performance expections is a very creative activity, and requires insight into market behaviour. Coincidentally most creative professionals are not technical: the people employed by industry to enhance products do not understand or realise the limitations of the embedded computer science (§4).

User interfaces are complex and problems in usability rarely show up immediately. By the time a user becomes dissatisfied with a product, Moore's Law 'ensures' new products are available. There is little need for manufacturers to make old products better if, by the time their limitations are discovered, users prefer a new product to a fixed older product. Thus Moore's Law encourages loss in design quality: manufacturers can assume customers prefer to upgrade to a faster or more featured product. In other words, provided the product lasts long enough — a few months — it need have no long-term usability features: this makes it easier to design, and to get away with poor quality design methods. Even if a product fails completely, consumers would probably rather pay to upgrade than wait for it to be repaired.

Moore's Law encourages bad computer science. Your *current* programs will run 25% faster in six months with you doing absolutely nothing. In particular because memory gets cheaper, programs can get bigger. Moore's Law removes pressure to improve program quality: programmers can always postpone addressing bloat, particularly if developers are using the latest machines. Suppose, by way of example, a program is so complex that it will take 3 years to run. If its programmers *do nothing* for 18 months, and then start, cheaper, faster computers will be available. The program would still finish on the same date. The programmers could have 18 months idling, and still achieve the same goal. Whatever you want to do, you may as well ignore problems to start with.

Enthusiasts of Moore's Law talk huge numbers. Processor speed has improved by s, memory has increased by m, and, *wow* the product has twice as many zeros! But if we have more memory we need a faster processor just to take advantage of it in the same time. If there is memory not accessed, it is being used inefficiently; thus improvement might be better quantified by the ratio m/s, which assumes the computer is *used* to capacity in both speed and memory. Talking huge products $s \times m$, while superficially impressive, ignores usage.

Since the ratio m/s, whose dimension is time, is more-or-less constant, it explains why, despite the naïve view of Moore's Law, computers *still* take as long to boot! (Hence Machrone's Law: the computer you want always costs \$5000.) It comes down to a business issue, we want computers that are more powerful than our competitors, and we can afford to finance just so much to get the leverage. Despite Moore's Law, users still take approximately the same sort of time to perform meaningful tasks.

If the growth of Moore's Law meant anything real for users, we would already have had a *huge* computer revolution: doubling every 18 months is a factor of a million after 30 years. In comparison, with a meagre factor of only 200, more realistically maybe as little as 10, social life has been transformed by transport.[1] The transformation transport has made within such a small factor suggests that the terafactors for computers are measuring something of little significance to humans. Humans do want to get to other places and they do want to move manufactured goods around the world, but they do not need computers to have more memory *per se* or higher density processors. What they do want is to get tasks done faster and better. Computers are not doing that with anything like the spectacular improvements the proponents of Moore's Law would like us to believe, except in a very few specialised areas [10].

Human activities generally have not been speeded up, and certainly not speeded up by spectacular factors. The growth described by Moore's Law focuses on technology not human impact, and encourages industry to postpone worrying about usability, and encourages consumers to buy to keep up with changing fashions — all despite causing continual waste (§7). Ironically market churn, a corrollary of Moore's Law, encourages a low standard for the computer science of everyday things, which in turn makes a causualty of usability.

[1] Before the Industrial Revolution people moved between the speed of walking or of a horse, say 5km/h to bursts at 70km/h; today you can (unsafely and expensively) travel on land at up to 1000km/h, though most transport is undertaken by lorries travelling around 110km/h (much less if we average over the journey rather than take maximum legal speeds).

3 Consumer (in)action

The automobile industry was changing rapidly in the 1960s, and in some ways the problems of computer science forty years later reflect it. In the 1960s, some cars were badly designed and unsafe to drive. As Ralph Nader exposed [14, 22], the prevailing cultural assumption was that drivers had accidents, and therefore drivers were responsible for the behaviour of cars. If an accident is "driver error" it is not the manufacturer's problem, but is, say, a human factors or contextual problem. If a parked car rolls down a hill (an example from [14]), the driver should have applied the parking brake properly, they should have turned the wheels so the car would roll against the kerb, and so on — they should be better trained, the handbrake should have better affordance ... all perfectly constructive observations. That the car has been badly engineered and has an unsafe parking brake is thereby disguised. Similar remarks could be made about "pilot error" and that generic excuse, the "human factor."

Thus we are persuaded that usability problems are human problems. If users *have* usability problems, they should learn how to use things properly. Our society is certainly a complex place, and people do need to be technology-literate because that is how the world is. But this practical response can be used as (yet another) excuse to make systems arbitrary and over-complex because users will take it upon themselves to learn how to use them. Indeed, manufacturers often commodify their learning material, thus making further profit by providing systems that require additional training!

The prioritisation of safety in car design came about because of consumer action, stimulated by Nader's damning exposés. Currently consumers of everyday things are unaware usability problems are avoidable. People working in industry are themselves consumers of everyday things — they may even be employed *because* of their eager following of technological fashion; they are not immune from the cultural assumptions. Although consumers could be focussed [8] towards usability, the main driver for change will be from technologists. Thus, the rest of this paper is aimed at computer scientists, arguing

Figure 2: The Casio SL-300LC. Note the highly-visible self-advertising memory keys in the middle of the top row of buttons.

(*i*) the computer science of everyday things is embarassingly bad (*ii*) the computer science is easy to fix, and to do so would have valuable benefits — achieving environmental as well as usability benefits.

4 Buggy everyday things

There are many gadgets representing bad computer science, but most are so bad that they are impossible to describe clearly! Perhaps the best example, because it is in principle straight forward, pervasive yet inexcusably bad, is the handheld calculator.

Hand held calculators are a mature technology. Calculators have well-defined requirements, of accuracy and performance and so on. There have been many generations of calculator designs, and the manufacturers have had many opportunities to step their production to fix problems. Research into calculator user interfaces has a twenty-year background [25, 29]. The only limitation on calculator quality is manufacturer inclination.

Casio is the leading manufacturer of hand held calculators. Two of their basic models are the SL-300LC and the MC-100, which look very similar. The SL-300LC is shown in Figure 2.

- Pressing the 10 keys $\boxed{\text{C/AC}}$100+17.5% (e.g., to calculate 17.5% tax on £100) on the SL-

300LC gets 117.50. The MC-100, with *exactly* the same keystrokes, gets 121.21212. **Two confusable calculators, made by the same manufacturer, work in very different ways.**

Since the [%] key is unpredictable (from calculator to calculator, and even in different contexts in the same calculator), users soon avoid using it, thereby 'solving' the usability problem with a work around. With effort they succeed, and because of their 'success' it becomes hard to question the overall appropriateness of the device for their tasks.

Both calculators have memories, which (appear to) function identically. The button [MRC] recalls the stored number and displays it, but pressed twice in succession it sets the memory to zero. The button [M+] adds the displayed number to memory, and [M−] subtracts from the memory.

- We can assume that the memory is there for a purpose: to store numbers, and perhaps especially to store numbers that have been calculated (since other numbers are likely to be easily available). How can a number calculated and displayed be stored in memory? It is very hard to work out what to do.[2] **Despite high-visibility, deceptively simple keys, memory is very difficult to use.**

Arguably, memory should save paper and help users do sums more reliably. Yet most users (especially those that need calculators) would need a scrap of paper to work out how to avoid using paper to write down the number! Evidently memory is not provided to make the calculator more usable but is a feature to increase p (see Figure 1).

Casio has been making calculators for a long time, and the two calculators are not "new" in any way. It is not obvious how Casio can justify either the differences or the curious features shared by both calculators. Neither comes with user manuals or other information that warn of or reveal any problems.

Calculators are impressive, especially if a prepared salesman shows you them going through some calculations. It is *possible* to demonstrate the memory in action, apparently working. Only some critical thought would determine that it is a very peculiar, confusing and unreliable feature.

4.1 More examples

A cynical reader might take the Casio calculators as specially selected (but see [29]). This section gives a brief selection of problems to illustrate the diversity of problems. All should have been easily detectable, and all are easily avoidable in principle. See [9, 18, 20] for examples of usability problems with safety-critical devices, where there is an obligation improve usability.

The Sony television KV-M1421U has a remote control, the RM-694. Not only are their colour, number and layout of buttons different, but the user interfaces are unrelated: they work in completely different ways — which is obvious, e.g., from their statecharts [27]. One concludes either Sony never specified the user interface, or did not worry about the gratuitous differences.

The Panasonic *Genius* microwave cooker has a clock. The user interface allows the clock to be set to any number, `0000` to `9999`. Despite the impression that it may be a 24 hour clock, it only runs when set to a valid 12 hour time. Nothing gives any indication why the clock will not run when it is set to a time such as `2230` [23, 31].

The Nokia 5110 mobile phone has a menu-driven user interface providing access to 'all' the phone's functions (see §5). However some functions, such as keypad lock, are not in the menu structure. Thus a user without the phone's manual may be unable to find such functions. There is no reason why all functions should not be in the main menu.

The De Longhi Pinguino [5] air conditioner cannot be used without the user manual. There are delays between pressing buttons and things happening, so a

[2] Pressing [M+] adds the display to memory, and only makes the memory equal the display if it already contains zero, which it may not. To make the memory zero, you have to press [MRC] twice, but pressing it the first time sets the display to the memory, which loses the number you wanted to store. There is no easy soluton, and no solution avoids the risk of calculator overflow. (If you need paper to work it out, why not use the paper for the memory?!)

user might press a button twice (to really do something) but this resets the mode of the device before it has even started. When the device is running, its "on" light is off. A front panel light labelled "memo control" is described in the manual as meaning that there is a three minute wait (if it is flashing) or that the air conditioner is in cooling mode (if it is on continuously). It is possible to press the buttons on the front panel to get all the lights to come on in interesting patterns; this is presumably a test mode (or perhaps a salesroom eye-catching feature), but the manual does not describe it.

The JVC HRD580-EK video recorder has the opposite problem [21]. It has features that cannot be used *with* the user manual. The manual has textual descriptions of complex operations (such as tuning in the VCR to UHF channels). Reading these sections is time consuming, yet the corresponding features on the VCR have short time-outs. So by reading the manual while trying to perform the operations described ensures the VCR will time-out: then the user will be reading the wrong part of the manual to explain what the VCR is doing! What the user is reading becomes counter-productive. (One solution to the system/manual synchronisation problem is for the device itself to present instructions to the user: since it 'knows' what state it is in, it can always present the appropriate sections of the manual [1].)

There is not space here to analyse these assorted criticisms further. Each unusable system is unusable in its own way; the faults are too varied to easily bring into clear comparison. (See [26] for a systematic analysis of a long list of usability problems with a single product.) Faults persist because nobody has conceptual leverage on the problems: manufacturers employ production processes that result in incoherent usability issues; nobody understands the results. The problems are 'obviously' not computer science, since it is silent on the mess.

5 Computer science applied

We need conceptual leverage not on problems, interesting as they may be, but on solutions. User interfaces might be improved by better computer science, but can improvements be achieved systematically? In this section we outline a computer science appraisal of the Nokia 5110 mobile phone's user interface. We provide evidence that applying elementary computer science will improve user interfaces.

5.1 More efficiency, based on Nokia's design

Full details of the example discussed in this section can be found in [28].

The Nokia 5110 handset organises most of its functions in a tree, which is presented to the user as a menu. Clearly the design assumption is the user will search the tree to find functions to activate. Computer scientists are familiar with search, and indeed with search trees.

A Huffman tree is the most efficient way to organise a search tree (given that there is a fixed set of keys) [28]. We could simply determine the average cost per function and compare it with Nokia's design, but the so-called cost of knowledge graph, itself introduced by psychologists to evaluate user interfaces, shows increasing efficiency the longer it is used for the Huffman over Nokia's original design; after 500 key presses, a user of the Huffman design is twice as efficient.

It can be pointed out that Nokia's design may have virtues other than trying to minimise keystrokes; certainly, a Huffman tree is optimal precisely because it has no other purpose than to be efficient. However all Nokia functions have a shortcut code, which (as it is so-named) is presumably to provide a brief alternative. A Huffman tree lends itself to shortcuts; moreover, since Nokia do not use their shortcut codes very efficiently, we can provide Nokia's original (whatever its merits) and a non-overlapping efficient code: this gains the best of both worlds (whatever the 'best' of the Nokia world is supposed to be). Nokia's shortcuts have an average access cost of 3.64 keystrokes; the combined approach has an average cost of 2.87 keystrokes.

Such improvements can be made by applying purely routine computer science to user interface design. The next section illustrates what we can do when more imagination is brought into play.

5.2 More efficiency, based on a novel design

Full details of the example discussed in this section can be found in [12].

Most keys on the Nokia have letters on them, and just as they are used for entering names (e.g., SMS messages) they could be used for searching for functions. Reference to any algorithms book will find numerous ways of making good use of keys for searching. Here we explore hash codes.

Given a function name, such as ***Call Waiting***, we find its hash code by using the Nokia's existing keypad. Thus, the first two letters of ***Call Waiting***, "C" and "A," are on the 2 key, and the third and fourth letters are on 5. Thus the hash code of ***Call Waiting*** starts 2255... As a user enters a hash code, the new design displays (in a scrolling list) all functions that match the hash code. To disambiguate codes, the user can either use up and down arrows (as in the conventional Nokia design), or enter more digits. If the user presses the C key, the last digit is deleted, and the hash code looked up again and the display is refreshed.

Marsden built a simulation of the original Nokia and the new design, and compared their use. (Both designs used the same screen image.)

Of the experimental scenarios completed on the new design, users achieved an average 9.54 key presses, in comparison to the Nokia design where 16.52 key presses were required. This was a strongly significant result ($p < 0.001$) with users requiring approximately 7 fewer key presses, on average, to access the functions on the hash code design.

The overall mean times (not key press counts) for the hash code phone was 33.42 seconds, compared to 42.02 seconds for the original. The improvement, though, was not consistent enough from person to person for a statistically significant difference to be asserted.

A computer scientist would wonder why the alphabetic labels are not more sensibly organised. Of the 12 available keys, only 9 are used, and the allocation of letters to keys does not reflect frequency of use. There are international standards of letter allocation, but as phones differ, they cannot be an important consideration — and the silliness of the international standards merely begs the question why the standards bodies do not ask computer science questions either.

5.3 Other benefits

Both examples above rely on a specification of the Nokia device. Once the device is specified, it is easy to devise and analyse many improvements.

The inconsistencies in the actual user interface of the Nokia suggest that Nokia do not use a professional computer science approach to user interface implementation.[3] If a more-or-less formal approach had been used, inconsistencies would have been *hard* to implement, whereas with an informal approach inconsistencies are *easy*, if not an expected consequence! It is arguable that consistency makes devices easier to use (outside of games); certainly consistency makes user manuals easier to write and shorter. In fact user manuals can be generated, at least in part, automatically [30] — then, not only can user manuals be used concurrently early in the design process, but they can have guarantees (if desired) that they are complete and correct.

6 Programming the user interface

Computer scientists routinely design computer programs to be efficient and correct. We need to see the user as at least as important as a computer, and the user interface as a program for the user. User interface design then becomes a matter of "programming the user." Many problems in user interface problems then come down to failing to provide the user with a correct and efficient user interface algorithm.

It is obvious that computers and human users both 'run' programs (in languages such as CSP [7] the differences that might seem obvious in languages like Java disappear). In particular, users deserve as much

[3] Two examples: short cut codes are not displayed uniformly from function to function; the C key behaves differently (unnecessarily) in different functions. In ***Phone book***, while entering a number, C deletes digits, whereas in ***Clock***, again while entering a number, C goes up a level in the menu hierarchy. In ***Clock***, then, it is not possible to correct numbers: *all* of a number's digits have to be entered correctly first time.

attention in the design of *their* programs (i.e., the user interfaces, user manuals, user models) as the computers get in their programs.

That the user interface therefore needs to be computable is obvious [24] — but this is a principle that seems to be ignored in almost all interfaces! Even such simple tasks as determining a sequence of steps moving a cursor from one screen position to another in Microsoft Word are undecidable.

Once user interfaces are seen in this light, then many ideas from computer science can be applied. This paper showed how familiar algorithms (for search) can be applied to user interfaces; elsewhere we have explored more general ideas, like applying declarative programming to user interfaces [25].

Given that many user interface problems can be attributed to bad computer science, conventional empirical usability work is misplaced, at least while it tries to understand or make user interfaces look better without changing the underlying way they are implemented.

There are numerous reasons why change will be difficult, including:

1. Some human factors researchers working in usability are explicitly trying to find ways of camoflaguing problems in user interfaces [19].

2. Most human factors trained professionals do not understand computer science, and there is a professional division between the two subjects.

3. The accurate computer science description of typical user interfaces is too hard to undertake, creating the impression that computer science cannot speak on the topic. In contrast, user problems are relatively easily observed and measured. This makes human factors salient, but it describes symptoms of computer science problems rather than helps understand or avoid their cause.

4. Because all user interfaces involve people, very similar empirical methods can be applied very widely. In contrast almost all computational approaches are particular — for example, discrete systems methods would be inappropriate for virtual reality; techniques for speech are inappropriate for graphics — and much harder to generalise. In comparison to human factors experts, computer scientists seem to know relatively little about user interfaces in general.

5. User interface design is done by industrial designers, again who do not understand computer science.

6. Because of the market demand, practising computer scientists, even ones not working in user interfaces, are trained to a low standard.

7. Manufacturers put their best programmers onto tasks that must work correctly. In a mobile phone, say, these will include battery management and radio communications. Customers will notice quickly if the phone does not work at all; but most customers will blame themselves when they find the user interface tricky. Thus manufacturers put their worst programmers onto user interfaces.

8. Computer scientists who develop a product are familiar with its user interface. Their knowledge and experience will make it (and in general, all devices) much easier for them to use. Therefore computer scientists discount the evidence of usability studies.

9. Because it stimulates consumption, users are encouraged to blame themselves rather than the products they buy. Researchers, too, are consumers with similar stakes in the usability of the everyday things they own. It is unusual and depressing to question the design of products you have chosen to buy, especially if it brings into question the standards of your own profession!

7 Sustainability and ethics

With all the reasons against improving user interfaces, where will improvement come from?

Moore's Law apparently underwrites manufacturers' confidence in faster and more powerful processors for their products. Manufacturers can anticipate adding features to systems without seriously impacting their performance. Equally, they can anticipate users preferring to replace their current products with new ones — users prefer to replace rather than have inferior devices fixed. By the time users find problems with user interfaces, the market will have moved on and the users can be promised their problems will be fixed by new features.

The result of this is continual replacement. Industry benefits, and consumers willingly become fashion followers. Mobile phones, for example, are not sold on their effectiveness for communication but on their status value.

There is a serious consequence: what happens to all the obsolete devices? In the UK, 900 000 tonnes of mostly toxic waste electronics is buried annually in land fill sites. This is not sustainable. Some notable efforts have been made to recycle, for instance shipping computers to the third world — but this is merely shifting the toxic waste from one back yard to another. Manufacturers may be persuaded by legislation [4] to use more environmentally-friendly materials and processes, but this does not address the core problem.

The computer science of everyday things has been so bad for so long that we all take it for granted. It is time to design user interfaces so they are robust, so they can be upgraded. These are conventional computer science issues.

This section has made the point that ethical issues are relevant; but see [11], which shows that car radios are a widespread, unethical, example of dangerous user interfaces; and see [6], which discusses ethical design more generally.

8 Conclusions

We are caught up in a culture that accepts unusability. Competent computer scientists are part of it and, sadly, aloof from the user interface. Yet computer scientists routinely prove properties of their programs and worry about algorithms and efficiency: why not apply these sorts of concern to the user interface? A professional computer science of everyday things would help improve usability.

There is a computer science of everyday things. This paper showed that benefits, at least of usability and a better environment, will follow from applying it.

References

[1] M. A. Addison and H. Thimbleby. Intelligent adaptive assistance and its automatic generation. *Interacting with Computers*, Volume 8, Number 1, pages 51–68, 1996.

[2] J. S. Brown and P. Duguid. *The Social Life of Information*. Harvard Business School Press, 2000.

[3] C. M. Christensen. *The Innovator's Dilemma: When Technologies Cause Great Firms to Fail*. Harvard Business School Press, 1997.

[4] Commission of the European Communities. *Proposal for a Directive of the European Parliament and of the Council on Waste Electrical and Electronic Equipment*, Volume COM(2000) 347 provisional. 2000.

[5] de Longhi. *Pinguino Electronic eco, 573909/02.97*, 1997.

[6] P. Duquenoy and H. Thimbleby. Justice and design. In M. A. Sasse and C. Johnson (editors), *IFIP Conference Interact'99*, 1999.

[7] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International Series in Computer Science, 1985.

[8] M. Jones, G. Marsden and H. Thimbleby. Empowering users with usability certificates. In S. Turner and P. Turner (editors), *BCS Conference HCI 2000*, Volume II, pages 37–38, 2000.

[9] P. B. Ladkin. Analysis of a technical description of the Airbus A320 braking system. *High Integrity Systems*, Volume 4, Number 1, pages 331–349, 1995.

[10] T. Landauer. *The Trouble with Computers*. MIT Press, 1995.

[11] G. Marsden, P. Duquenoy and H. Thimbleby. Ethics and consumer electronics. In A. D'Atri, A. Marturano, S. Rogerson and T. W. Bynum (editors), *Fourth International Conference on the Social and Ethical Impacts of Information and Communication Technologies, Ethicomp'99*, 1999.

[12] G. Marsden, H. W. Thimbleby, M. Jones and P. Gillary. Successful user interface design from efficient computer algorithms. In *ACM Conference CHI'2000, Extended Abstracts*, pages 181–182, 2000.

[13] G. Moore. *Nanometres and Gigabucks — Moore on Moore's Law.* University Video Corporation Distinguished Lecture, 1996. `http://www.uvc.com`.

[14] R. Nader. *Unsafe at Any Speed.* Pocket Books, 1965.

[15] B. A. Nardi and V. L. O'Day. *Information Ecologies: Using Technology with Heart.* MIT Press, 1999.

[16] D. A. Norman. *The Psychology of Everyday Things.* Basic Books, 1988.

[17] D. A. Norman. *The Invisible Computer.* MIT Press, 1998.

[18] E. Palmer. 'Oops, it didn't arm,' a case study of two automation surprises. In R. S. Jensen and L. A. Rakovan (editors), *Eighth International Symposium on Aviation Psychology*, pages 227–232, 1995.

[19] B. Reeves and C. Nass. *The Media Equation.* Cambridge University Press, 1996.

[20] N. B. Sarter and D. D. Woods. How in the world did we ever get into that mode? Mode error and awareness in supervisory control. *Human Factors*, Volume 37, Number 1, pages 5–19, 1995.

[21] H. Thimbleby. Can anyone work the video? *New Scientist*, Volume 129, Number 1757, pages 48–51, 1991.

[22] H. Thimbleby. Computer literacy and usability standards? In C. D. Evans, B. L. Meek and R. S. Walker (editors), *User Needs in Information Technology Standards*. Butterworth-Heinemann, 1993.

[23] H. Thimbleby. The frustrations of a push-button world. In *Encyclopædia Britannica Yearbook of Science and the Future*, pages 202–219. Encyclopædia Britannica Inc., 1993.

[24] H. Thimbleby. Treat people like computers? In A. Edwards (editor), *Extraordinary People and Human-Computer Interaction*, pages 283–295. Cambridge University Press, 1995.

[25] H. Thimbleby. A new calculator and why it is necessary. *Computer Journal*, Volume 38, Number 6, pages 418–433, 1996.

[26] H. Thimbleby. Design for a fax. *Personal Technologies*, Volume 1, Number 2, pages 101–117, 1997.

[27] H. Thimbleby. Visualising the potential of interactive systems. In *Tenth IEEE International Conference on Image Analysis and Processing, ICIAP'99*, pages 670–677, 1999.

[28] H. Thimbleby. Analysis and simulation of user interfaces. In S. McDonald, Y. Waern and G. Cockton (editors), *BCS Conference HCI 2000*, Volume XIV, pages 221–237, 2000.

[29] H. Thimbleby. Calculators are needlessly bad. *International Journal of Human-Computer Studies*, Volume 52, Number 6, pages 1031–1069, 2000.

[30] H. Thimbleby and P. B. Ladkin. A proper explanation when you need one. In M. A. R. Kirby, A. J. Dix and J. E. Finlay (editors), *BCS Conference HCI'95*, Volume X, pages 107–118, 1995.

[31] I. H. Witten and H. Thimbleby. User modelling as machine identification: New methods for HCI. In H. R. Hartson and D. Hix (editors), *Advances in Human-Computer Interaction*, Volume IV, pages 58–86. Ablex, 1993.