A future for usability engineering

HAROLD THIMBLEBY

Middlesex University

This paper reviews the “unusability culture” and its symptoms. For usability engineering to have an effective future it must take advantage of new design tools and techniques. A particular approach is discussed in detail. It is argued that the approach, based on “interactive design assistants” is effective, and can be used to make radical improvements to design processes and to usability research. The approach reduces barriers between participants in usability: between designers, programmers, technical authors, users. By running on the World Wide Web, the assistants would open access to usability engineering, and increase public awareness and participation in design. The approach is argued to contribute to a new future for usability engineering.

Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User / Machine Systems, H.5.2 [**User Interfaces**]: Evaluation / methodology — Interaction styles — Theory and methods — Training, help, and documentation — User interface management systems (UIMS)

General Terms: Design, Documentation, Human Factors, Theory, Verification

Additional Key Words and Phrases: Usability engineering, Software Engineering

1. INTRODUCTION

Interactive systems are ubiquitous and economically important. They are bought in quantity, improve productivity, and, as the landfill sites full of discarded electronics show, they are also disposed of in quantity. What is seen as “the solution” soon becomes obsolete, and another solution is offered and taken.

This paper argues that current interactive systems are badly designed, but they are not so badly designed as to be unusable. Indeed, because they promise so many advantages for users, it is, at best, iconoclastic for this paper to suggest all is not well! Thus, in support of our critical position, we review some symptoms of the current design culture. We need some powerful ways of levering out of the present apathy; this paper then provides a practical way towards a solution — a future for usability engineering. Our contribution is based around a design tool, which requires some engineering sophistication to build, but once built will be a powerful tool to promote usability in many ways.

This paper focuses on small consumer goods, typically those with large markets and little direct contact between designers and users, though many of its arguments apply to larger systems, most especially to the interactive subsystems (e.g., control panels) of larger systems.

2. THE USABILITY ENGINEERING TENSION

A wide variety of skills are recruited to the design process that leads to the final use of an interactive device. Marketing skills, business skills, engineering skills, production skills, usability skills, distribution skills — all, and more, are required to deliver a product to users. Distinctive to interactive devices is the need to meet appropriate usability criteria, and *usability engineering* is the discipline that aims to improve usability [Nielsen, 1993].

Unfortunately there is a tension between the two approaches of ‘usability’ and ‘engineering.’ Software engineers are rarely interested in usability; usability experts are rarely interested in programming. To a software engineer’s mind, that some usability work can be undertaken with polystyrene mock-ups with imaginary interaction only serves to emphasise their trivialised role. To a human factors specialist the mathematics of software engineering is far removed — if not completely divorced — from human issues. There is animosity between the disciplines: both demand specialist competences, and each can accuse the other of not understanding

Author’s address: Computing Science, Middlesex University, London, N11 2NQ, GB.
Tel: +44 181 362 6061; Fax: +44 181 362 6411; email: harold@mdx.ac.uk.

crucial issues. The two disciplines seem to have nothing to contribute to each other, despite working towards the same end product.

There is a usability engineering tension.

It seems self-evident that usability should be user-centred. (This is a central message of much work in usability, well-expressed by [Landauer, 1995].) Programmers make out that programming is an easy task, and that programming user interfaces is easy too; if so user-centredness is “all there is” of substance left to contribute to design requirements. But our view is that software engineering is by no means easy,[1] though it is deceptively easy to make it look easy. Systems that are well-engineered are few and far between, and the design quality of most interactive products is low and out of control (§3,4). Therefore to improve usability *it seems* one has to concentrate on users and post-design problems. This is a practical response to the problem; but a strategic response would be to *also* improve design practice.

Figure 1 represents the “usability engineering tension.” The diagram also corresponds with the physics of use: users are on the outside, there is an interface, and the engineering is on the inside of the device.

Clearly, some design problems *are* engineering problems (e.g., the system doesn’t work), and clearly some problems *are* usability problems (e.g., the buttons are too small). Most are better seen as a mix. However, by promoting user-centred usability, the engineering focus shrinks, so the possible contribution of engineering to usability is diminished. Ironically this makes iterating system design harder, because less theoretical engineering development occurs so the system is not robust enough to modify reliably. In turn, specious problems accidentally reinforce the need for user-centredness, and so the focus shrinks further.

This paper proposes a tool that reduces the tension. In fact, it will reduce tension between users, human factors specialists, technical authors, software engineers, interface designers and hardware designers … it does this by providing a complete solution to a manageable, and core, part of the design process.

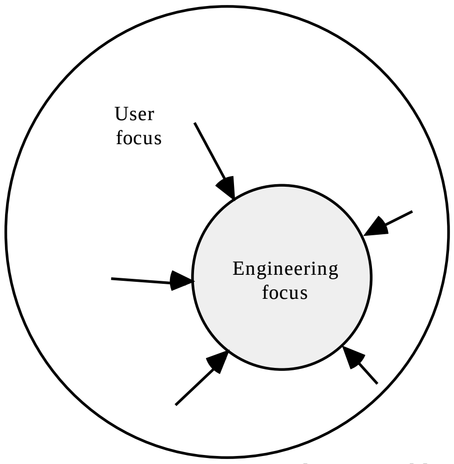


Fig 1. The usability engineering tension: interactive system design problems are perceived as usability or engineering problems in varying proportion. When usability alone is emphasised, engineering shrinks and is less able to solve or proactively avoid design problems. When engineering is emphasised, usability problems are discovered too late to fix.

3. SYMPTOMS OF POOR DESIGN

Is an improvement in usability engineering required? I believe so, and this section recruits a selection of examples to make the point. The usability problems we list here simply should not arise; none of the examples here are innovative products. The usability research has been available for years; fundamental concepts like predictability, observability of internal state, and so on, were well-known in the 1980s [Thimbleby, 1980; Smith & Mosier, 1984; Norman & Draper, 1986], popular works such as [Norman, 1988], numerous textbooks such as [Shneiderman, 1998; Dix, Finlay, Abowd & Beale, 1998] (both popular enough to continue into new editions), and more recent research [Buchner & Funke, 1993; Abowd, Wang & Monk, 1995; Thimbleby, 1994] — to say nothing of the conferences, the journals and the wealth of material

1 Few professional programmers understand much of the standard undergraduate programming textbook [Knuth, 1997].

on the World Wide Web. Somehow this knowledge — whether old or contemporary — is not transferring to practice. Usability engineering does not seem to be working.

We examine the user interfaces to products made by leading manufacturers. I am not trying to review the products, and I am not trying to criticise the manufacturers. These manufacturers and their mass-produced products are chosen because they have leading designs and they reflect the state of the art. Because their products are mass produced and very widely distributed, even small usability problems have an impact on humanity that can be considered scaled up by the number of users affected.

3.1. Casio calculators

Hand held calculators are an example of a mature technology. Basic 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 known problems. The only limitations on calculators are the manufacturers' imagination and skill, perhaps modified by any marketing pressures. Research into calculator user interfaces has a twenty-year background [Young, 1981]. More recent discussion can be found in [Thimbleby, 1996].

Casio is the leading manufacturer of hand held calculators. Two of their basic models are the SL-300LC and the MC-100. These calculators look very similar.

— A simple Value Added Tax calculation on 100 pounds is 100+17.5%. Pressing the 9 keys 100+17.5% on the Casio MC100 gets 121.21212, while on the very similar-looking Casio SL300LC the same key presses get 117.50.

  Since the % is unpredictable, users soon avoid using it, thereby 'solving' the usability problem with their own work around. Thus with hard effort they succeed, and because of their investment in solving problems it becomes harder 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: namely, to store numbers, and perhaps especially to store numbers that have been calculated (since other numbers are likely to be known or easily available). Given that the calculators have a memory, then, how can a number calculated and displayed be stored in the memory? (Pressing M+ presupposes the memory contains zero. And to make the memory zero, you have to press MRC twice, but doing that sets the display to the memory losing the number we wanted to store!)

Thus a market leader, Casio, makes two similar calculators that work in subtly different ways, and both proclaim features that are 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! I conclude that the memory is a feature that is not provided to make the calculator more usable.

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 reveal any problems.

Any calculator, and the Casio ones in particular, can be demonstrated. They are impressive, especially if a salesman shows you them going through some typical (but unsophisticated) calculations. It is possible to demonstrate the memory in action, and only some critical thought would determine that it is a very weak feature.

3.2. Sony televisions

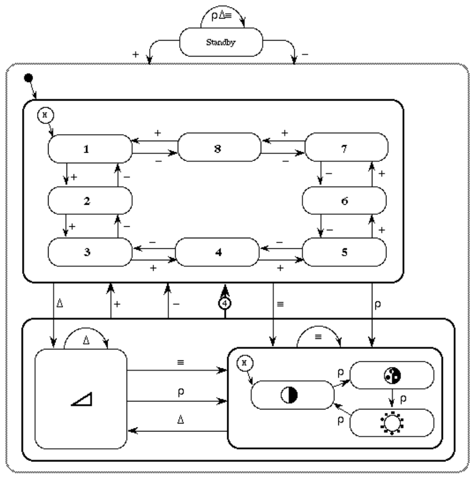
Sony is a leading manufacturer of consumer electronics. We consider the Sony KV-M1421U type TV together with its remote control, the RM-694.

Figure 2 shows two statecharts specifying how the user interacts with each device, the TV or the remote control. We do not need to understand statecharts [Harel, 1987; Harel & Politi, 1998] to see that the devices are very different; even the corresponding buttons do different things.

Some features can be done on the TV alone, some can be done on the remote control alone, some features available on both are done differently on each device. It would appear Sony never visualised the user interaction, certainly not using anything like statecharts.

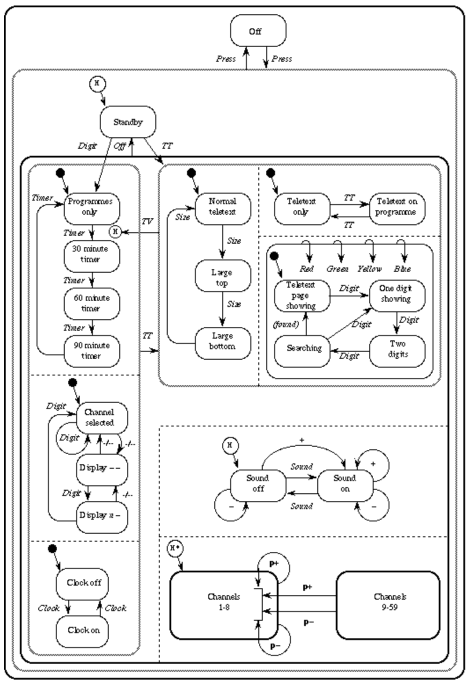
Although there is only one application, namely the TV, the user has two user interfaces to understand, with their own independent rules. If the user manual explained how to use the TV properly, then having two different interfaces for it would require a user manual of double the thickness. Moreover if a user becomes skilled with one interface then their skill is of little use with the other: there is little scope for transfer of skills from one interface to the other. A user who loses their remote control may find the TV interface unfamiliar.

An interesting usability issue can be seen in the television statechart (on the left in Figure 2): the TV has a class of states (shown at bottom of the statechart) that cannot be quit *except* by doing nothing until the time-out. Trying to get out — by pressing buttons — keeps the TV resetting the timer.



Note. ≡ means either of the ≡+ or ≡− buttons;
Δ means either of the Δ+ or Δ− buttons

The TV statechart



The remote control statechart

Fig. 2. Statecharts for a Sony TV and its remote control.

3.3. Canon cameras

Canon is a leading manufacturer of cameras. The Canon EOS500 is a popular automatic SLR (single lens reflex) camera. Unlike the Casio calculators or the Sony TV, the EOS500 camera has a substantial user manual. Not only does the manual explain the operation of the camera, it provides photographic hints and tips. It also exhibits informal usability insights — evidently ones the company are not exploiting.

The EOS500 user manual warns users that leaving the camera switched on is a problem. There is an explicit warning in the manual on page 10:

> "When the camera is not in use, please set the command dial to 'L'. When the camera is placed in a bag, this prevents the possibility of objects hitting the shutter button, continually activating the shutter and draining the battery." [Canon, 1993]

Thus Canon evidently *know* that the lack of an automatic switch-off is a problem! In short they ask the user to switch the camera off — rather than designing it so that it switches itself off (for instance, after a time out).

Leaving the camera accidentally switched on is a recipe for wasting film; the next example from the manual is another example of potential film wastage built-in to the design:

> "If you remove a film part-way, the next film loaded will continue to rewind. To prevent this press the shutter button before loading a new film." [Canon, 1993]

Since this problem is documented in Canon's own literature, we can be sure Canon is aware of the usability problems, but somehow fails to fix them before the product is marketed.

Three years later Canon released an updated version of the EOS500, the EOS500N. The user manual for this camera phrases the same problem thus:

> "If the film is removed from the camera in midroll without being rewound and then a new roll of film is loaded, the new roll (film leader) will only be rewound into the cartridge. To prevent this, close the camera back and press the shutter button completely before loading a new roll of film." [Canon, 1996]

It seems the manual writers have now discovered that as well as pressing the shutter button, the camera back must be shut too (it would probably be open if you were changing a film). It doesn't seem like the EOS500N camera designers read the EOS500's manual themselves.

3.4. Some small examples

A cynical reader might take the previous examples (§3.1–3.3) as specially selected. This section gives a brief selection of problems that illustrate the diversity of problems (see [Ladkin, 1995; Palmer, 1995; Sarter & Woods, 1995] for examples of similar problems with safety-critical devices). All should have been easily detectable, and all are easily avoidable in principle.

The Panasonic 'Genius' microwave cooker has a clock. The user interface allows the clock to be set to any number, 0 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 [Witten & Thimbleby, 1993; Thimbleby, 1993b].

The Nokia 2110 mobile phone has a menu-driven user interface, providing access to 'all' the phone's functions [Nokia, 1997]. 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 some functions should not be in both the menu and also have 'quick access' short cuts.)

The De Longhi Pinguino [De Longhi, 1997] air conditioner is a typical mysterious interactive system that cannot be used without the user manual. There are delays between pressing buttons and things happening, so a user might press a button twice (to *really* do something) but this changes the mode of the device back before it has even started. The confusion is not helped that when the device is running, its 'on' light is off. The front panel light labelled "memo control" is described in the manual as meaning either 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 it may be a salesroom eye-catching feature), but the manual does not describe it. The manual provides a picture of the front panel of the device, with arrows pointing at each light and button, and providing helpful explanations of what they do. Arguably this information might have been put on the front panel of the device; for without the user manual, the front panel is inscrutable.

The JVC HRD580-EK VCR has the opposite problem [Thimbleby, 1991a, 1991b]. 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 itself have short time-outs. So by reading the manual while trying to perform the operations described more-or-less ensures that the VCR will time-out and then the user will be reading the wrong part of the manual to explain what the VCR is doing! This is a special (perhaps rather perverse) case of the system-manual synchronisation problem. To be useful, the section of the user manual must be synchronised with the state of the interactive device. If the section the user is reading gets out of synchrony with the system, what the user is reading will be counter-productive. (One solution to the 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 only appropriate sections of the manual information. We shall see below how to do this systematically.)

There is not space here to analyse these assorted criticisms. But see [Thimbleby, 1997] which gives a systematic analysis of a long list of usability problems with a BT fax, the DF200. Each unusable system is unusable in its own way, and the faults are too varied to easily bring into clear comparison. And this is the reason why faults persist: usability engineers have no conceptual leverage on the problems, and manufacturers employ production processes that result in incoherent usability issues, and nobody understands the results. While the best usability engineers can do is to pick off one problem at a time, progress will be falling further behind the advance of technology.

4. UNUSABILITY AS A CULTURE

An economics response to the pervasive unusability in consumer goods would be to say that the market is fragmented, and therefore consumers can be exploited. Turn of the century "push button" technology recapitulates the mid-century issues with the automobile markets. By the 1960s problems with car usability were widely recognised, but manufacturers argued — at first successfully — that drivers had accidents, so drivers were the problem. Ralph Nader intercepted correspondence between insurers that made it clear the industry expected users (drivers) to sort out their own problems, for instance by becoming better trained to drive and handle the cars [Nader, 1965]. On the other hand, as Nader argued, some of the drivers' problems with cars arose from poor engineering. Even experts were unable to handle some unstable models.

One might argue, by analogy, that our current culture of accepting usability problems is a reflection of manufacturers shifting the focus from usability engineering to consumerism. It might be said, "There is no problem with usability, because new models are being brought out that solve all the problems." The trouble is, this is a cliché that has had a run of decades: everyday, we are promised, today's problems are fixable by tomorrow's developments. The industry is no longer in the position that it can claim that the technology is new or is stabilising. There are some serious, deeply entrenched problems.

Problems users have are classified as the user's problems. Thus users believe *they* make mistakes (even though poor design encourages them to make mistakes) and therefore they are responsible. They should buy upgrades, training, or perhaps get their children to use the systems. Never is the problem seen to lie with the technology! Always the problem is something the user can apparently buy themselves out of, thus giving more profit to the manufacturers. Manufacturers charge users for helplines and bug fixes; some manufacturers provide certification processes — that create a hierarchy of skills, that consumers pay to climb. There is little incentive for design to improve when users are so keen paying repeatedly for bad design.

But possibly the car industry was different. The Ford Motor Company's 1958 Edsel failed to sell; whereas Tamagotchis do sell despite being an almost perfect parody of well-designed systems! Tamagotchis, small hand held interactive children's toys, represent the unusability culture rather well: their purpose in life is to be difficult to use. They have to be cared for. Cults of experts gain esteem from becoming knowledgeable about them. Some people (especially young people) thrive on gadgets, thus making the losers feel that unusability is their fault.

In short, interactive systems are unlikely to achieve all intended goals (but they achieve enough intended goals most of the time), and they support and are promoted by a social structure that rationalises their continued production and consumption.

(Further analysis of the "unusability culture" can be found in [Landauer, 1995; Norman 1998; Postman, 1992; Thimbleby, 1999b].)

4.1. The Year 2000 bug

The Year 2000 bug, a term covering a variety of issues, has been the largest-scale (usability) engineering problem ever to face humanity. It presents a severe usability problem: devices fail to work, and become unusable.

It arises from unprofessional practice: people failed to specify requirements and to follow through on verification, and systems have been used outside their specifications. The poor practice includes bad programming, bad design, bad diagnostic procedures, bad documentation, and so on.

The lack of care for users implicit in software warranties is well known (Thimbleby, 1990a), but the Y2k bug makes the presumption of "usability is the user's problem" stark. With the Y2k bug manufacturers typically charge customers for fixing problems, entirely of the manufacturer's own making. In some cases, this charging occurred several times as the manufacturers repeatedly upgraded their badly-engineered systems.

The Y2k problem has been widely seen as a unique problem that needed fixing, rather than a significant example of a much deeper problem: that complex systems are badly designed, and that users are expected to sort out problems for themselves.

This curious attitude would not work in any other area. If a car was badly designed and had serious functionality or usability problems, the manufacturer would recall it.

Why is this? Problems with cars are usually reproducible. If the manufacturer has failed in some area of design, then all cars are affected by the same problem. The "single problem" is easy to define, and therefore it is reproducible by all users. An announcement in the media will draw all users' attention to the problem, all users can reproduce the problem, and make the same complaint. Thus manufacturers are put under pressure to fix the undeniable problem.

It is plausible that when usability problems are not reproducible or not understood, they are made out to be the users' problems. Because interactive systems are often built by low quality programmers, systems are not well defined. Because they are not well defined, their behaviour is inscrutable, and they interact unpredictably with other systems. So nobody can reproduce problems. Therefore problems are not the fault of the design, but must be the user's fault.

Our technical proposal (below) will enable usability problems to be reproducible, thus attacking this aspect of the culture of unusability.

4.2. Unusability culture affects experts

Professionals working in usability engineering itself are not immune to the effects of the unusability culture. It is very hard if not impossible to work within usability without having a personal stake in the field. Some of us use Macintoshes, some of us use Unix, some PCs. Whichever we use, we live in a community of users of the same sort of system — many things we do reinforce the wisdom of being in this community. Other people in the same community are more helpful to us (of course!) and they have solutions we can covet to help us, whereas people working with other systems do not know the answers to our particular problems, and they are not using solutions that we would covet. There are even lively magazines and other fora that emphasise the value of each system to its own community — and often in contrast to the comparative disadvantage of the alternatives.

Likewise, and on a wider scale, we all have video recorders, mobile phones, calculators, and have everyday experience of various sorts of gadgets. Often, then, we have a stake in a specific solution. Other sorts of solution are of no personal interest, because they do not help us use the particular gadgets we use.

As an academic discipline, usability engineering progresses by shared use of refereed literature. Unfortunately, one of the consequences of the fast-paced development of interactive technology is that for any academic development in usability, there is almost certainly a commercial product that excels some aspect of it — reason enough not to publish it as it is not original. What seems close-to-market is rarely "academic." So the research literature tends to avoid work that is relevant to manufacturers of interactive consumer devices. Yet clearly manufacturers do overcome technical problems to get the devices to work: they do stay abreast of the research in engineering; they do use development tools to design and build the devices, but these tools do not embed any usability engineering.

4.3. Lack of theory

Usability engineering suffers from a lack of powerful theories — or, more precisely, a lack of widely accepted theories that many people other than their originators can exploit. It has been shown that when there are no consensual theories, problems are dismissed as coincidences, curiosities, or as "that's just how it is" [Lightman & Gingerich, 1992]. Continental drift, for instance, had been proposed as early as 1800 to explain the close fit of Africa's and South America's shapes, but the idea was dismissed until the theory of plate tectonics was developed in the 1960s. Both evidence and theory, an explanation of the evidence, are required for problems to be taken seriously. Usability seems analogous: lack of theory — lack of effective insights into design — lead to usability issues being dismissed.

Usability engineering sometimes seems to confuse descriptions for theories. Many design tools (user interface management systems, UIMS [Olsen, 1991]) enable designers to construct realistic interactive systems very easily, but they do not make the properties of the systems easy to uncover. Typically, they use powerful scripting languages — so almost any interactive feature can be programmed or simulated using multimedia effects. But there is little underlying semantics that can be reasoned about. In many scripting languages, even simple questions such as "are all states accessible from everywhere" are not answerable in general. This is a fundamental limitation: any Turing Complete programming language (e.g., any half-decent scripting language) has properties that are undecidable. In fact, essential usability problems are undecidable. A theory-driven approach to usability engineering has to start from what is provable and see what can then be achieved; this is the opposite approach from the usual one, of seeing what can be achieved and not worrying about the theory! (The system we propose can do all we claim for it using only finite state automata, and — being finitary — more properties are decidable.)

Usability engineering needs to develop theories and systems for the reliable design of complex technology, to clarify problems and make them visible at early points in the design process, where they can be managed, analysed and avoided. Unfortunately, usability is very complex. A complete definition of the user interface of something as simple as a vending machine is at the limits of researchers — vending machines are regular examples in the software engineering literature and in the usability literature, so they clearly present challenging problems for both specialities. Researchers must either concentrate on idealised systems, or gloss the difficulties in knowing what the systems are. Left with idealisations or without theories, product designers make arbitrary decisions, which in turn are hard to theorise!

5. SOLUTIONS

What can be done? The market is fragmented, usability engineering standards are dismal, and nobody believes there is a serious problem that can't be bought out of, and certainly not one that can be fixed by investment in usability engineering!

Granted that unusability is a besetting problem, these are the priorities:

— *Defragmenting consumers.* Allow users to compare systems. Enable consumer groups to expose problems objectively.

— *Demystifying usability engineering.* In particular opening access to the programming of interaction. Usability problems (and successes) must be made reproducible, and must be separated from proprietary systems.

— *Supporting effective usability-informed production processes,* including iterative design and concurrent design.

— *Defragmenting experts.* Provide points of view (e.g., theories or tools) that transcend specific systems and cultural barriers between specialisms. The usability engineering tension must be dissolved! Providing theories that experts from diverse specialisms agree on will be hard, but embedding theories inside tools will make them more accessible to non-specialists. Whatever solution is chosen, it must be complete, and be a common tool between the design specialisms.

This paper's goal is to show that these are achievable.

Common to all of the usability problems are six core roles, which we pick out because they represent where design information is stored (or should be stored):

C The initial design concept, from which everything in the design process flows.

S The system specification. This typically resides in the heads of the system engineers, and ultimately becomes represented in the device's electronics and firmware.

A The appearance specification. This initially resides in the heads of the industrial designers, and ultimately becomes embodied in the design of the interactive system's case, buttons, and layout.

D The device itself, representing the system specification and the appearance specification to the user.

M The user manual of the system, as well as other documentation, such as material for helpdesks. This is typically written by technical authors who may or may not have access to any engineering specification of the device. There are also other design documents, such as cognitive walkthroughs, scenarios, design rationale and so forth; for the sake of simplicity we shall treat them together.

U The user(s) of the interactive system.

These roles are related in sequence in Figure 3. The diagram will be used to develop a line of argument, rather than to contrast with, say, waterfall or spiral models. The figure does not show, for example, that a technical author interprets the device, and might themselves interact with it, in order to write the manual.

The arrows represent information flow that ultimately inform the user's interaction with the device. The engineering specification and the industrial design of the device takes place first, and from this the device is constructed. The user interacts with the device after it is built. Some manufacturers believe that technical authors should not "take advantage" of things users do not know; it is conventional wisdom [Jackson, 1983] to argue that until the system is actually built, nobody knows what it is going to be anyway — so user manuals should wait as long as possible in the production process. This reactive approach for M is true even in variants informed by usability considerations [Sutcliffe & McDermott, 1991]!

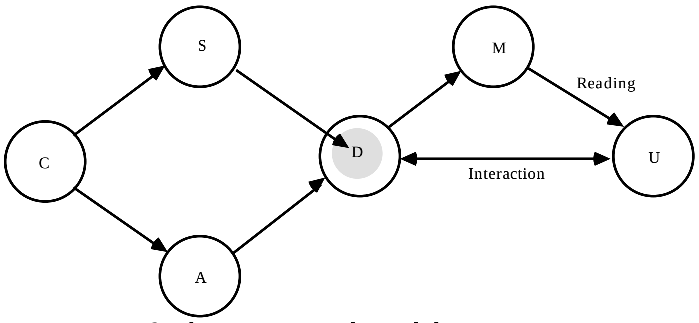


Fig. 3. The conventional serial design process.

The steps, as one proceeds from the left of the diagram to the right, are each expensive, slow, and largely *ad hoc* manual processes. Manual processes are unreliable; André (1989) argues a railway accident would have been avoided by automating manual generation.

Any change to the concept C really requires a new design: change to the specification is usually too difficult to take back to the concept to revise it. In short, iterative design — the recommended cycle of development, testing, and refinement of a design — is costly and slow. The time it takes to design is comparable with the production life of the product: there is no point doing iterative design if the resources required could instead be used for producing a new product altogether.

In Figure 4, we have supposed that as much as possible of the process has been automated, by providing a design tool (not shown explicitly in the diagram). The engineering specification S and the appearance specification A have been combined into a joint specification SA. In principle, this is not a problem — SA might for instance be the pictures and scripts of a multimedia design tool (of which there are many commercial examples). From SA, both the internal engineering of the device D can be obtained, as can its outward appearance. This much is routine practice in industry today, though often the scripts will be converted partly by hand into something appropriate to build the electronics embedded in the final device. But if the specification SA is sufficient to build the complete system, then it must be (almost) sufficient to build the user manual M — although strictly possible, this step is rarely automated because of its complexity, and the need to make M a high quality document.

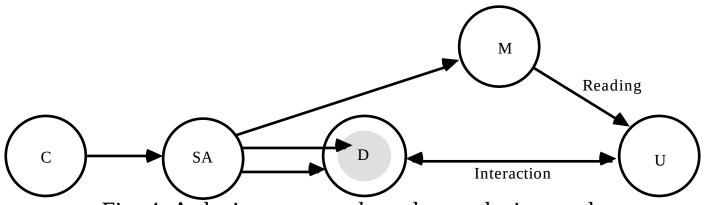


Fig. 4. A design process based on a design tool.

If SA contains enough to generate M, then this information could have been fed into D too, then D could generate an interactive version of M. For example, VCRs with on-screen help (where the TV they are connected to shows instructions) are the sort of system that would benefit from this facility. (So-called on-screen help is little more than form-filling, rather than information that helps the user achieve goals. We will see how more useful help can be achieved, below.)

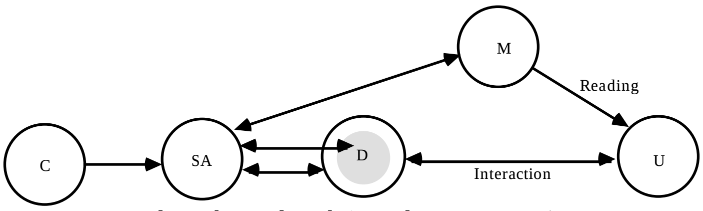


Fig. 5. Extending the tool with ‘equal opportunity’ interaction.

If the arrows emanating from SA represent fully automated processes, and hence fast and efficient processes, then we can do better. For example, if the manual M is generated from SA, then any text in M must have some representation in SA. One could allow the technical authors to edit M, to improve the style or to clarify automatically generated explanations. These changes to M could be carried back into the appropriate places in SA. The technical authors’ changes could *stay* represented in SA. Then if a design was to be modified, the technical authors’ improvements would be carried into the new device’s user manual. In fact, the technical authors could do more than simply use SA as a store for their text: since other design specialists use SA as well, the shared representation could support dialogue between the designer team. A technical author might annotate the specification — some part of the design might be really awkward to explain, or might require the user to do acrobatics — and these annotations could be related back to appropriate parts of SA, where they would be visible to, say, engineers who could adjust the design to fix the problems identified by the explanation.

We could also allow changes to the device D to be carried back into the specification SA. Indeed, in many multimedia development systems, the designers can edit a running simulation of D — standard commercial tools make the editing and modification of designs so easy that there is hardly any need to distinguish between S and D for the purposes of prototyping.

The change from Figure 4 to Figure 5 was by making some arrows double-headed; this is an application of the user interface design principle of *equal opportunity* [Runciman & Thimbleby, 1986; Thimbleby, 1990b]: that nothing is considered exclusively input or output to the design tool. All information has equal opportunity to be either input or output in the process. Thus, with suitable tool support, the manual can be viewed either ‘in’ SA or ‘in’ M — the conventional serial one-after-the-other approach to design is superseded. Since equal opportunity tool support means providing design input can be done as easily in M, D, or SA the design process is concurrent. Every participant in the design process — engineers, technical authors, interface designers, etc. — themselves have an equal opportunity to contribute, and can do so at any time.

A particular use of equal opportunity is *literate using* [Thimbleby, 1986]. Design documents M no doubt contain examples of use (user manuals will have examples; design documents will have use cases, etc.). If these are purely written manually, they are not guaranteed (e.g., when the device specification changes). If they are written specially — by description, perhaps by demonstration — then the tool itself can generate the examples (i.e., the tool takes fragments of the document as input and splices the output — hence the equal opportunity). This is particularly useful during design, when many documents may be generated, discussed and revised rapidly. For example, a cognitive walkthrough [Nielsen, 1994] would be automatically updated, and places where the examples fail could automatically be drawn to the designers’ attention [Libes, 1995].

Finally, in Figure 6, we go a stage further. Since the tool’s features are now extensive, the objects have been relabelled with a * to emphasise the more flexible and extensive roles possible in the equal opportunity design process. Taking full advantage of the equal opportunity, we continue to blur the distinctions between the specification, the manual and the device, for the tool makes the transformations — originally done by hand — so efficient that they can be considered free and reversible. Moreover, since the tool can so efficiently transform between manual, specification and device, we no longer really require a separate concept C. Indeed, even in many commercial environments, C never exists materially as a separate stage — standard multimedia production tools are “so good” that C can be fully or almost fully developed entirely within the tool.

The box on the left of Figure 6 represents all preproduction activities; when a design is ready for production, the prototype manual M* is transformed into a fixed manual M for delivery with the fixed (physical) device D. Because changes to the specification now have negligible knock on costs *inside* the box on the left of the diagram, iterative design can really be achieved practically.

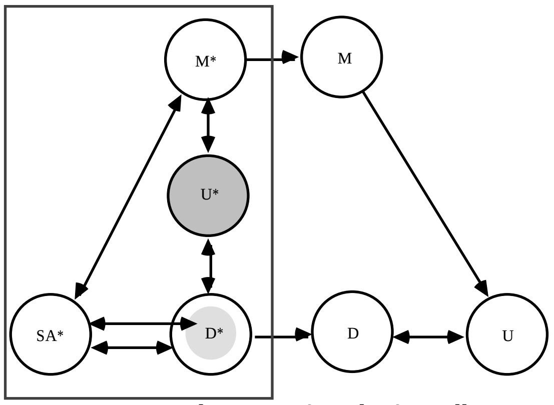


Fig. 6. Sketch of new design processes exploiting IDA tools. As well as static documentation, D will contain interactive help.

Iterative design within the box is practical: therefore there is a new role, that of the test users, indicated by U*. The test users help evaluate the design (by interacting with a prototype, using manuals M*, by participating in focus groups, and so on). If the tool is suitably designed, the device D* could be simulated not just on a computer screen, but could be made available on the Internet (i.e., the World Wide Web), and hence the users U* could be anywhere in the world. It is now possible to use the design tool to help target the device for multicultural use, and the manual (or manuals) for multilingual use.

It is helpful at this stage to introduce some terminology: the design tool that supports the activities within the box in Figure 6 (and the initial steps of the arrows leading out of the box) represents an *interactive design assistant*, or IDA. An IDA is a user interface management system (UIMS) but has distinctive advantages.

With an IDA there is no binary distinction between prototype and full-scale implementation, since the IDA makes everything work, and it provides continuity from sketch to complete system (since everything works at every stage). So when Nielsen writes,

> "One should not start full-scale implementation efforts based on early user interface designs. Instead, early usability evaluation can be based on prototypes of the final systems that can be developed much more cheaply, and which can thus be changed many times until a better understanding of the user interface has been achieved." [Nielsen, 1997, p93]

he is assuming the old distinction between prototypes and completed designs. But in an IDA the final design is as easy to change as the first, and this is true even if, say, the technical authors have been developing M* concurrently. Because M* (and M) can be derived with little effort, users can understand the design as it evolves.

To summarise Figures 3 to 6: the first figure showed the serial linkage between the roles in conventional interactive systems design, but as more tool support is used, there is a progression through the figures. Figure 4 represents standard industry practice in 1999. Proprietary tool support that is available makes the transition to Figure 5 plausible, and in Figure 6 the assumptions about the design tool have been fully extended. The design tool, now called an IDA, has collapsed the separate preproduction stages of design into a single concept where iterative design can be achieved almost at no cost, and where it becomes feasible for all parts of the design team — industrial designers, technical authors, engineering technicians — to collaborate concurrently. They can work independently or collaboratively.

Of course, this ideal supposes it is practical to build such an IDA! Let us postpone that question and first see what advantages may be had for the new design process that is made possible by an IDA.

5.1. What can an IDA do?

The IDA idea fits conventional approaches to usability; it is effectively recommended by [Gould, Boies and Lewis, 1991] as one of their four design principles — to integrate design, so that all aspects of usability (user interface, help systems, training, documentation) should evolve concurrently. As [Shneiderman, 1998] says, "the advantages of specialised user-interface software tools for designers and software engineers are large. They include an order-of-magnitude increase in productivity, shorter development schedules, support for expert reviews and usability testing, ease in making changes and ensuring consistency, better management control, and reduced training necessary for designers." These are attainable goals for an IDA.

What is the difference between an IDA and a UIMS? An IDA is a user interface management system, certainly, but most UIMS have tried, as their priority, to handle the realities of complex devices, they try to be realistic. Their goals are expressiveness, rapid prototyping and sophisticated user interface features [Myers, 1993]. In contrast, the style of an IDA is that it establishes relations between SA*, D* and M*, and the rigorous enforcement of these relations and the exploitation of the two-way (equal opportunity) guarantees of relatedness are essential. In other words, the style of IDA-based work is to ask, "what can be designed well?" rather than, "how can we support the design of what is being done already?" For, as we saw in §3, existing practice needs changing: a UIMS that merely made existing design easier would not be worthwhile. It may be, then, that an IDA cannot do some activities that existing design tools can, because the IDA approach will not compromise. (Whether such activities are generally worthwhile is another question; most likely we would find that IDAs would improve usability simply by making certain sorts of weak design impossible.)

Properly implemented and designed, an IDA would provide *guarantees*. Thus the user manual would accurately describe the interactive device, all modes and features of the device would be documented, and so on. Thus an IDA would be ideal for designing devices intended for safety and mission critical applications. The guarantees would be invariants: that is, at all times the various components of the IDA (device, manuals, specification) would be consistent.

The radical effect of equal opportunity in the design of an IDA and the consequential immediacy of swapping between different representations cannot be over-emphasised. The designer can swap between different representations — and see the user interface more from the user's point of view. The user can swap between different representations — and can be better understood by designers. The technical author can swap between different representations — and can identify bugs in the design, rather than just use hindsight to warn about them. [Alexander, 1965] makes powerful arguments that designers tend to choose representations that make design easy (for example, they choose tree structures), but this rarely leads to designs that are easy to use (because trees separate functions). By reducing the gap between design and use — the equal opportunity in the simulated device means that it can be designed as it is used — an IDA ameliorates the problem Alexander identifies as otherwise pervasive.

Currently many design competitions (such as the UK Royal Society of Arts's Student Design Awards) heavily emphasise industrial design rather than interaction design — appearance rather than function. This is partly a cultural specialism, but also a practical response to the present difficulty of doing both sorts of design at once. But an IDA would make design easier so that it would be possible to do both. Industrial design without interaction would be as unthinkable as interaction without appearance. In short, an IDA made available to design students would be a start to promoting good usability engineering practice (such as collecting empirical evaluation data) amongst those students.

An IDA would be a typical Web application, and run on almost any platform. Currently, people working on one design project are rarely in a position to share their insights and design criticisms with others because their design tools are not portable, they may be expensive and proprietary. Researchers are rarely funded well-enough to use commercial tools (and some tools are commercial secrets).

An IDA running simulations on the Web would vastly extend the number of people who could access and understand designs. If the IDA itself was sufficiently easy to use, the design process itself would become more open. One of the most important aspects of an IDA would be to increase the number of people who understand design of interactive systems. For example a simple IDA could be used for school use in design projects — and its analysis features could help tutors evaluate designs for their usability. Some of those students would become designers and in turn produce systems commercially. Ideally the early exposure to clear usability issues would lead to long-term better design: this is essential for tackling the unusability culture at its roots.

By widening access to design out of the specialist communities, an IDA would help increase the public's understanding of usability. It is possible that manufacturers would put up interactive Web sites that present their devices, and give users an opportunity to test them against their own requirements. Consumer groups would create generic tasks (such as "setting the video recorder") so that consumers could try out suites of tests on various devices.

There would be widespread uptake and real testing of design ideas, as well as far more effective collaboration and support of related work, whether at school, industrial or research levels. Making usability engineering issues visible (and providing features that highlight them) would promote wider public understanding of the issues behind usability.

5.2. Making disruption acceptable

An IDA offers so many advantages for improving design, that using it effectively would itself change design practice. An IDA is a *disruptive technology* [Christensen, 1997]. But alleged advantages are not sufficient. An IDA would itself have to be usable, and fit for its purpose. But more: there are four main areas an IDA would be useful: for manufacturers (as businesses), for designers, for consumers, and for usability research. The IDA would have to make different sorts of claims to the different communities, and it would have to do forcefully enough to help break the unusability culture:

— *For business*, to be adopted an IDA would have to improve business processes; it would have to speed the time from design to product, or it would have to speed the design iteration cycle (supposing iterative design is being used). An IDA would have to speed the design process. An IDA would have to improve the market of (good) interactive systems. Businesses need to stay in the market to reap the benefit of better usability, and this presupposes they are selling products. An IDA would have to reduce after-sales costs. By improving usability, by improving manuals, and by helping provide better task-fit, an IDA should help reduce the numbers of users who buy the wrong products, or who buy products they do not fully understand. Thus, an IDA would show its success in the long term, as companies reduce the level of after-sales support. In turn, this would allow companies to invest in better current designs.

— *For designers*, an IDA would have to show itself to be flexible and better than current practice. It would have to seemlessly integrate its different parts without imposing collaboration if designers, for some reason, do not want to break down conventional disciplinary or organisational barriers. In product design, there are (at least) two sorts of designer: those who design the engineering, and those who design the packaging. A successful IDA would have to provide benefits for these two groups working together — it would have to allow them to work independently if that was what they wished, yet it would have to provide added value that neither could get without it.

— *For consumers*, an IDA would have to be usable (and sometimes *more* usable itself than the products it simulates and evaluates). Manufacturers will not improve their products when their products are consumed by an uncritical public, and the public will remain uncritical while there are no standards on which to base effective criticism. Ironically this could mean that helping consumers would make an IDA less attractive to manufacturers. When a customer telephones a helpdesk with a usability query, the problem should be reproducible and understood by the experts; help would then be more appropriate for the user.

— *For researchers*, the IDA approach presents research issues in several distinct areas: (*i*) assuming the IDA works fully, then new design approaches, team design, usability, manual design; (*ii*) assuming the IDA approach needs development as such, then it presents research issues in software engineering, interactive system specification, natural language generation; (*iii*) and all existing areas of usability research, since they all apply to the systems that the IDA helps design.

Researchers might find creating a successful IDA a suitable task in its own right; however that presupposes they are convinced an IDA is worthwhile. Obvious value-added that an IDA provides are logs and records of design processes and interaction, and moreover ones that can be exchanged with other researchers. An effective IDA should help build a usability research community.

6. THE NECESSARY BACKGROUND TO AN IDA

The IDA is a grand plan with huge promises, *but it relies on being possible*! An IDA has to meet several technical requirements:

— It must use a tractable specification, so that things (manuals, hardware designs, usability analysis) can be derived reliably.

— It must use an analytic specification, so that guarantees can be made and proved. This is essential if an IDA is to be used for safety or mission critical design.

— It must use an executable specification so that device simulations can be run.

— It must use a specification that can be annotated easily, for instance to track editorial changes to user manuals, colour designs for device facias, and so forth.

— For the aspects of design process it covers, it must be complete. Its workings must treat all participants in the design process equally and openly.

— It must be multi-platform, with a non-proprietary public architecture. It must be easy to use, yet extensible. This means it must be self-documenting.

What can do all this? Many systems can almost do this: for example RapidPLUS [Emultek, 1998], Simulink/Stateflow [Mathworks, 1998], Statemate [Harel, Lachover, Naamad, Pnueli, Politi, Sherman, Shtull-Trauring & Trakhtenbrot, 1990; Harel & Politi, 1998], and our own prototype IDA, Hyperdoc. Hyperdoc shows that an IDA-like approach is possible and effective. Hyperdoc in fact is an old system and has limitations. After describing Hyperdoc, next, we shall show that its limitations are easily surmountable using today's technologies.

6.1. Hyperdoc — a 'proto-IDA'

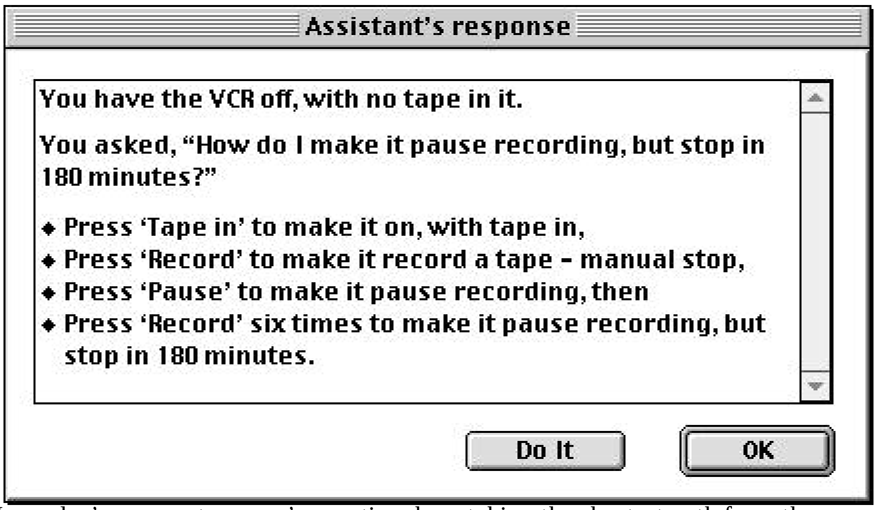
Hyperdoc [Thimbleby, 1993a; Thimbleby & Addison, 1996; Thimbleby, 1999c] animates models of interactive devices, generates user manuals (interactive, Web, and paper), and supports device analysis. It can use a scanned image of a real device, and buttons and indicator lights can be positioned over the image. They then work as on the real device (Hyperdoc can be used with a touch screen, but it is normally used with a mouse in the usual 'desktop computer' way).

Hyperdoc has several modes of use, in addition to simulating devices. The appearance of the device, such as the positioning of buttons, is done by direct manipulation. A spreadsheet-like interface allows template text for user manuals to be entered, and for the specification of the device to be edited, as well as to define user tasks. Since Hyperdoc can simulate interfaces, it "knows" about the behaviour of the device, and can use the manual templates to construct various sorts of manual — Hyperdoc provides built-in manual formats, which it generates in HTML.

Hyperdoc can also provide a variety of interactive help based on the templates while a simulation is being run. (Thus avoiding the synchronisation problem that paper user manuals have: in Hyperdoc the interactive help is based on what the user is actually doing — there is no chance that the system is doing one thing, and the user reading a different part of the manual.)

The specification (SA* in the discussion above) is stored as text so that it can be manipulated outside of Hyperdoc without restriction. Hyperdoc stores it as a *Mathematica* expression so it can be manipulated by *Mathematica* itself. (*Mathematica* is a powerful symbolic mathematics program [Wolfram, 1991], and is used with Hyperdoc to help analyse user interfaces and can easily be programmed to generate types of user manual that Hyperdoc cannot generate directly.)

Many things can be done. For example, a user can benefit from Hyperdoc's knowledge of the device and effectively interact in new ways with it. Suppose the VCR is off, and the user wants to get it going and stop recording in 180 minutes — normally this sort of task would be tedious to do. Instead, the user can ask Hyperdoc to find the best way (under any of various assumptions) of achieving the task. A typical question and answer is shown in Figure 7.



You have the VCR off, with no tape in it.

You asked, "How do I make it pause recording, but stop in 180 minutes?"

- Press 'Tape in' to make it on, with tape in,
- Press 'Record' to make it record a tape - manual stop,
- Press 'Pause' to make it pause recording, then
- Press 'Record' six times to make it pause recording, but stop in 180 minutes.

Fig. 7. Hyperdoc's answer to a user's question, here taking the shortest path from the current state. Hyperdoc can also run through the necessary steps, as a demonstration for the user. (Although the action verb can be changed, sadly every action in a simulation has to use the same word — here 'press.')

As user's work with the device simulation, Hyperdoc records their activities. This can be useful for usability analysis; it can also be used to view the use from the device's point of view — some states or button presses may not have been exercised much by the user's actions. If so, should they be redesigned? Such state and action distributions are productive to analyse (e.g., with Markov models [Thimbleby, 1999a; Cairns, Jones & Thimbleby, submitted]).

Various methods can be used to structure the manual — to make it more usable for particular sorts of task. Below is a manual generated by a greedy approach that maximises the size of related clusters in the manual. It isn't obviously a very good way of writing general-purpose manuals, but it shows how representations of manuals can give the designer — if not in this case the user — insight into device design. Hyperdoc can also save manuals as (hypertext) Web pages, so that "pressing buttons" in the manual then interactively takes the user directly to the relevant parts of the manual.

For the following (fast forward; pause playing; fast backward):
> You can press **Play** to play a tape:
> If you are playing a tape, but have paused it, additionally you may:
>> Press **Forward** to fast forward
>> Press **Rewind** to rewind a tape
>
> For the following (fast forward; fast backward) you can press:
>> **Pause** to pause playing a tape.

Other forms of manual can be generated by using weighting functions in *Mathematica*. For example:

— Clustering states by common out-arrows leads to a statechart-like user manual. Doing this usually identifies the on / off button as crucial to all operation!

— Manuals can be generated that say how to get to states regardless of the starting state. (A manual generated for a Hinari microwave cooker showed that the quickest way of reaching any state from anywhere *never* used the Clock button.)

Hyperdoc can be set up with a collection of pre-defined tasks. The simplest sort of task might be to end up with the device in a particular state, but more interesting ones can be specified like "stop the tape without switching the VCR off." Hyperdoc uses a weighted shortest path algorithm to find the best ways of achieving the goals, which are specified as a combination of invariants (e.g., don't switch off) and termination conditions (e.g., stop the tape).

If each button press is weighted equally, Hyperdoc gives the solution that takes the least number of button presses. But by weighting path costs, Hyperdoc can produce adaptive help for users. For example, if every time the user presses a button, the cost of that action is reduced, then Hyperdoc will tend to give answers the user is familiar with. Or conversely, if the costs were *increased*, Hyperdoc would give novel answers — which could be useful for a user training system. Or the weights can be established by experts (e.g., that any action that involves switching the VCR off mid-task is not recommended) and then the answers Hyperdoc would produce would reflect the experts' best practice.

User interfaces can be compared quantitatively (Table I, Figure 8).

Table I. Comparison of two similarly positioned video recorders.

	JVC model	Philips model
Mean distance between states	3.73	1.93
Proportion of actions that behave like "undo"	35.8%	55%
Average cost of recovering from an overrun error (the number would be 1.0 if there was an undo button)	3.76	2.02
Worst overrun recovery cost	11	6
Cost of correcting a random one-press error	2.44	2.1
Number of rooted spanning trees (how many manual structures are there to choose from?)	6×10^6	4×10^{10}
Probability that a random button press does nothing	0.53	0.27
Length of a test sequence, to check the model is correct	462	341

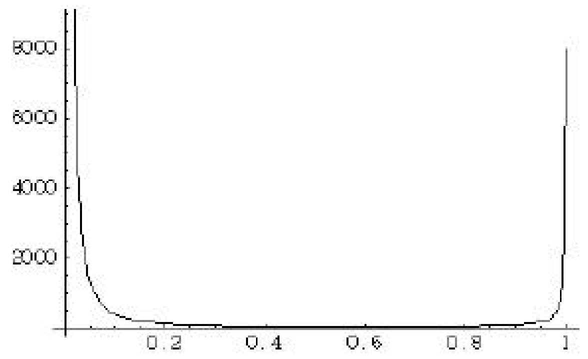


Fig. 8. Graph, from a Markov model, plotting cost of performing a task against probability of pressing the "time" button. The best performance is achieved when the "time" button is pressed with probability 0.594 — suggesting that it should be larger than the other 5 buttons on this device. From [Thimbleby, 1999].

More abstract analyses can also be made, for example several VCRs are unnecessarily complex, in the sense that they are not decomposable into simpler flow graphs [Thimbleby & Addison, 1994].[2] Such results beg interesting research questions: for example, does it matter from a usability point of view that a VCR is not decomposable to flow graphs? If experiments show it does, and perhaps in a quantifiable way, then the design of interactive systems could become much more reliable: a tool such as Hyperdoc would continuously indicate to the design team whether the interactive system design was decomposable. Designers would then use the tool's measures to guide their search for a good design.

Note that in all examples here, Hyperdoc knows nothing about VCRs or any other devices as such — it is a general purpose simulation system.

6.2. Extending Hyperdoc to be an IDA

Hyperdoc is merely a proto-IDA, and while showing that a simple technical approach can achieve a lot, it does not have a number of core IDA features.

— *Equal opportunity*. Hyperdoc allows the technical author to edit user manuals only in the specification SA. It does not support equal opportunity when generating user manuals. (So if a generated user manual was edited, Hyperdoc would not update the specification.) For example, the interactive help (Figure 7) is always output from Hyperdoc, there is no mode where a designer can edit the text *in context* to improve it. Editing user manuals directly, including structuring the manual into sections has been demonstrated elsewhere [Thimbleby & Ladkin, 1995].

— *Literate using*. Hyperdoc keeps different views of a device quite separate (except when they are stored in the *Mathematica* specification, but this is too technical to be much use!) Instead, information must be merged so that different workers can manage their

2 The basic constructs of structured programming are flow graphs: thus the result implies some interactive systems do not permit users to use 'structured thinking,' and are therefore unnecessarily difficult to understand.

material, but *automatically* stay up to date with other developments in the design. For example, cognitive walkthroughs describe the actions and effects on the device, so they should be automatically updated when the device specification changes. An example is given in [Ladkin & Thimbleby, 1997]: indeed this paper, complete with its examples was generated by the tool it deswcribes

— *Extensible*. Hyperdoc works with various support tools, notably *Mathematica*, but also graphics tools for drawing interfaces, text editors for manipulating specifications, and a Web browser for viewing user manuals. But all these tools are external to Hyperdoc: an IDA is integrated, and preferably component-based (so that the components can be separated and replaced as required by designers or experimenters). Making an IDA integrated is essential to making it easy to use (this requirement is compatible with being extensible, though it requires a self-documenting convention for the components).

— *Web ready*. Hyperdoc is a conventional application program. It is not available as an applet or cross-platform program.

In practice the worst aspect of using Hyperdoc is reverse engineering. It takes a long time to establish how interactive devices work, to enter their specification into Hyperdoc, and then to check them (though a shortest test sequence can be generated). Hyperdoc is tedious to use because — as we know — interactive devices are very hard to understand! But this is because Hyperdoc is a research tool; the solution to the reverse engineering problem is to make an IDA that is used by industry, then device specifications will not need to be reverse engineered.

6.3. A software architecture for an IDA

An IDA must have an open-architecture and run on the Web. It would therefore become a community project, rather like Linux, and by harnessing such co-operation could likely achieve the progress required.

Hyperdoc is implemented using an explicit finite state automaton model. This allows the correspondence between the simulation, the manual, and the specification to be maintained (and guaranteed) very easily, but it does have drawbacks. Finite state automata can get very large for even quite simple devices, and hence explicit automata are difficult to design with. One direct way to extend the power of the approach is to use logic, essentially so that classes of state are described by predicates. (Predicates specify properties clearly, which is also valuable for generating explanatory material.) In [Thimbleby & Ladkin, 1997] it was shown that propositional logic was sufficient to specify the user manuals for a subsystem of the Airbus A321 braking interface. Alternatively FSAs do not have to be explicit: they can be handled symbolically, by techniques such as binary-decision diagrams [Bryant, 1992] or in programming languages like Esterel [Halbwachs, 1993] that can be compiled into FSAs.

The core programming for IDA could start as a Java re-implementation and generalisation of Hyperdoc, extending it to annotate states with predicates (then it supports predicate action diagrams — PADs [Lamport, 1994]), combining it with a theorem prover (so it can model check the temporal logic PADs), and providing an extended manual editor, with audit feedback to the specification as in [Thimbleby & Ladkin, 1995].

Java interfaces will be defined so that the architecture is component-based, and does not depend on one engine (such as statecharts, Markov models or PADs). Many other approaches could be substituted and used in combination — moreover, with Java anyone anywhere can add such features for their own purposes or for sharing with the research community. There are various utilities to be added: such as for drawing statecharts, PADs and tables [Parnas & Lawton, 1998], for providing ‘movies’ to demonstrate systems; for collecting empirical data from Web use (with many users); and for life-cycle issues.

Java allows graphical realism or user manual features and other aspects to be ‘plugged in.’ Java’s interfaces would allow different suppliers to provide features for the core IDA tool. Thus IDA would be configurable to different design practices or to different production requirements. With such a flexible open-architecture, an IDA could be taken in different directions by different developers (say, in the quality of 3D imaging, or in the quality of natural language generation) — and yet provide extensions of benefit to all.

7. SAMPLE RESEARCH QUESTIONS

An IDA would raise numerous research questions, as well as make some existing research questions soluble and worth addressing. Here is a selection:

7.1. Design methodologies

A design methodology based around IDAs would be to generate user manuals automatically, and iterate design to minimise the size (or improve some other readability index) of the user manuals. Is this methodology effective? How would it be related to Carroll's powerful conception of minimal manuals [Carroll, 1990], when optimising minimal manuals could *drive* the design? While it might sound plausible, how would it work in practice? Could better methodologies be devised?

Another design methodology would be to use an IDA to promote closer participatory design. Arguments for designer-centred design have been promoted [Thimbleby, 1998b], since ultimately it is the designers who *make* new artefacts not users — but with an IDA, designers could make artefacts *with* users. Could users be designers? Would ubiquitous programmable devices (e.g., PDAs) lead to users designing personal interactive systems *for themselves* on a large scale, and how would an IDA help these users not make the same usability mistakes that today's designers are notorious for?

7.2. Benchmark task suites

Users will find an IDA useful if there are test suites and other ways of making comparisons between designs. Currently, with diversity in the market, there is little point in defining, say, a typical VCR user's activities — since the task suite would not work on more than one VCR, in which case it would not help users choose between designs! So, the research is to define task suites in ways that are helpful for users (i.e., related to lifestyle), yet generic enough to run in any IDA simulation. Once this is achieved, an IDA should be able to score designs against the task suites.

7.3. Diagrammatic notations

Statecharts are a diagrammatic notation that greatly simplify certain sorts of FSA structures, making them much more understandable. The interesting research question is whether good statechart diagrams make good usable systems. For example, would modifying a design so that its diagram was made clearer improve the usability of the device itself? More generally, is there a diagrammatic notation that has this very desirable property?

[Horrocks, 1998] is an elementary introduction to the use of statecharts in user interface design. (Curiously this book has no usability engineering in it.) It has a list of heuristics for helping designers: for example, "keep dependencies out of independent concurrent parts." These are all statechart-related heuristics. Do they help make user interfaces 'better,' and if so, in what ways for what tasks? The book, so interpreted, is a list of experimental challenges for usability research.

Predicate action diagrams, though being more general than statecharts, provide very considerable advantages for clear diagrammatic presentation. Would the greater expressiveness of PADs ensure that they are harder to use proactively in design, that is, would they conceal unwanted complexity?

7.4. Managing design rationale

We claimed that M could represent all manner of design documentation, not just user manuals. An IDA would have several documents types, for user manuals, service manuals, as well as for design documents such as scenarios and cognitive walkthroughs. Some design methods, particularly design rationale using gIBIS [Conklin & Yakemovic, 1991] and design space analysis [MacLean, Young, Bellotti & Moran, 1991] have their own structure (which is independent of the device structure). The research is this: an IDA is supposed to make interactive systems structure more manageable, so would its techniques make the structures of design rationale more manageable? What would a recursive IDA be like, that helps design its own documents in the same way that it helps design its own devices? This is the sort of idea that either has a very elegant result — and decimates the learning time for the IDA, because so much is done consistently — or results in a quirky system of little practical value. It is worth finding out.

7.5. Remote evaluation

An IDA can be used over the Web and can collect usage information. It would be ideal for collecting international and multicultural usage data. On the other hand, over-reliance on automatically collected data could skew evaluations — for example, only computer literate users would use an IDA in the first place. If email or other explicit feedback mechanisms were used, then the people who replied would be the ones with sufficient skill to manage sending email — users who experience interaction disasters would be under-represented in evaluation. The research question is how to handle this experimental bias.

8. CONCLUSIONS

We are caught up in a culture that accepts unusability: we have a stake in over-complex systems that we are over-proud of. This paper suggested an approach to help design for improved usability.

An interactive design assistant, or IDA, as conceived in this paper, would promote concurrent and iterative design, and it would automate many currently *ad hoc* practices. It would bring standardisation and evaluation into design. An IDA is an integrated tool, covering specification, documentation, design analysis and user interface simulation, and thus bringing together users, designers, technical authors, and researchers. In particular it would expose usability problems much earlier in the design process, so that there was a realistic hope of fixing them before production commitments had been engaged. But to escape from pervasive cultural forces is not easy, and more is required. Thus the concept of an IDA presented here is of an open-architecture system, widely accessible over the Web. Ideally it would become an international project, with international benefits. The implementation itself will pose interesting software engineering problems, but these could be solved by community effort, like the Linux project.

An IDA would be a visible and practical outcome of research in usability. It would both advance research, in design and usability, as well as make developments accessible to the commercial development world, where it is sorely needed. International design and usability competitions would be run using it. An easily accessible, formally-based, powerful IDA would contribute enormously to consensus and replicatable science.

Developments in Internet technology make the prospects of an IDA very exciting because it would make usability issues accessible from anywhere in the world, and because it could be designed so that collaborators anywhere would contribute components to it (for example, if someone wants to generate Spanish user manuals).

Finally, usability engineering should include an explicit “public understanding of science” element. This is valuable in its own right, but is valuable directly because usability engineering should have a vision about changing culture — inspiring, particularly, children to go on in life to identify and fix usability problems. A spin off would be a greater appreciation of the way in which our society depends on deeper concepts from usability engineering: it doesn’t just depend on the boxes called computers, but depends on programs (and documentation) being well-designed and appropriate for their tasks. By promoting public understanding of usability, in a way accessible even to children, then usability engineering would guarantee itself the best future possible.

ACKNOWLEDGEMENTS

Ann Blandford, Mark Addison, Paul Cairns, Matt Jones, Peter Ladkin, Gary Marsden, Yin Leng Theng, Ian Witten have all collaborated directly in the work described in this paper. Simon Peyton-Jones, David Pullinger, Prue Thimbleby and Peter Thomas made very helpful comments on versions of the paper.

An earlier version of parts of this paper were originally presented at the User Interfaces for Theorem Provers (UITP’98) conference.

REFERENCES

ABOWD, G., WANG, H. and MONK, A. 1995. “A Formal Technique for Automated Dialogue Development,” *Proceedings ACM Conference on Designing Interactive Systems’95*, 219–226, ACM Press.

ALEXANDER, C. 1965. “A city is not a tree,” *DESIGN*, **206**: 46–55.

ANDRÉ, J. 1989. “Can Structured Formatters Prevent Train Crashes?” *Electronic Publishing — Origination, Dissemination and Design*, **2**(3): 169–173.

BAUDRILLARD, J. 1998. *The Consumer Society*, SAGE Publications.

BUCHNER, A. AND FUNKE, J., 1993. “Finite-state Automata: Dynamic Task Environments in problem-solving research,” *Quarterly Journal of Experimental Psychology*, **46A**(1): 83–118.

CAIRNS, P., JONES, M. and THIMBLEBY, H. W. submitted, “Reusable Usability Analysis with Markov Models,” submitted *ACM TOCHI.*[*]

CANON INC., 1993. *EOS500/500QD Instructions*, part no. CT1-1102-006.

CANON INC., 1996. *EOS500N/500NQD Instructions*, part no. CT1-1111-000.

CARROLL, J. M. 1990. *The Nurnberg Funnel*, MIT Press.

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

CONKLIN, J. and BURGESS YAKEMOVIC, K. C. 1991. “A Process-oriented Approach to Design Rationale,” *Human-Computer Interaction*, **6**(3/4): 357–391.

DE LONGHI, 1997. *Pinguino Electronic eco*, 573909/02.97.

DIX, A. J., FINLAY, J., ABOWD, G. and BEALE, R. 1998. *Human-computer Interaction*, 2nd. ed., Prentice-Hall Europe.

EMULTEK, 1998. RapidPLUS^{TM}: An Emultek white paper. http://www.gkn-wsal.com

FLUKE CORPORATION, 1997. *Fluke 1997 Test Tools Catalog*, H0002EEN Rev. V 97/03.

GOODMANS, undated, VN6000 Operating Instructions, 97P9531403D02.

HALBWACHS, N. 1993. *Synchronous Programming of Reactive Systems*, Kluwer Academic Publishers.

HAREL, D. and POLITI, M. 1998. *Modeling Reactive Systems with Statecharts*, McGraw-Hill.
State of the Art, Holt, P O’B & Williams, N, editors, pp.222–243, Intellect Press.

HAREL, D., LACHOVER, H., NAAMAD, A., PNUELI, A., POLITI, M., SHERMAN, R., SHTULL-TRAURING, A. and TRAKHTENBROT, M. 1990. “STATEMATE: A Working Environment for the Development of Complex Reactive Systems,” *IEEE Transactions on Software Engineering*, **16**(4): 403–414.

HORROCKS, I. 1998. *Constructing the User Interface with Statecharts*, Addison Wesley Longman.

JACKSON, M. A. 1983. *System Development*, Prentice-Hall.

KNUTH, D. E. 1997, *The Art of Computer Programming*, 2nd. ed., **1**, Addison Wesley Longman.

LADKIN, P. B. 1995. “Analysis of a Technical Description of the Airbus A320 Braking System,” *High Integrity Systems*, **1**(4): 331–349.

LAMPORT, L. 1994. *TLA in Pictures*, Digital SRC Research Report, **127**.

LANDAUER, T. 1995. *The Trouble with Computers*, MIT Press.

LIBES, D. 1995. *Exploring Expect*, O’Reilly & Associates, Inc.

MACLEAN, A., YOUNG, R. M., BELLOTTI, W. M. E. and MORAN, T. P. 1991. “Questions, Options and Criteria: Elements of Design Space Analysis,” *Human-Computer Interaction*, **6**(3/4): 201–250.

MATHWORKS, INC., 1998. *Simulink*, http://www.mathworks.com

MYERS, B. A. 1993. “State of the Art in User Interface Software Tools,” *Advances in Human-Computer Interaction*, H. R. HARTSON & D. HIX, eds., **IV**, Lawrence Erlbaum.

NADER, R. 1965. *Unsafe at Any Speed*, Pocket Books.

NIELSEN, J. 1993. *Usability Engineering*, Academic Press.

NORMAN D. A. 1988. *The Psychology of Everyday Things*, Basic Books.

NORMAN D. A. 1998. *The Invisible Computer*, MIT Press.

NORMAN D. A. and DRAPER S. W. eds., 1986, *User Centered System Design*, Lawrence-Erlbaum.

NOKIA, 1997. *Nokia 2110*, 9350586.

OLSEN, JR. D. 1991. *User Interface Management Systems: Models and Algorithms*, Morgan Kaufmann.

[*] This reference will be replaced with a technical report if necessary.

PALMER, E. 1995. "'Oops, it didn't arm,' A Case Study of Two Automation Surprises," Jensen, R. S. & Rakovan, L. A., eds. Proceedings of the Eighth International Symposium on Aviation Psychology, 227–232. http://olias.arc.nasa.gov/~ev/OSU95_Oops/PalmerOops.html

POSTMAN, N. 1992. *Technopoly*, Vintage.

RUNCIMAN C. and THIMBLEBY, H. W. 1986. "Equal Opportunity Interactive Systems," *International Journal of Man-Machine Studies*, **25**(4): 439–451.

SARTER, N. B. AND WOODS, D. D. 1995. "How in the World Did We Ever Get into That Mode? Mode Error and Awareness in Supervisory Control," *Human Factors*, **37**(1): 5–19.

SHNEIDERMAN, B. 1998. *Designing the User Interface*, 3rd. ed., Addison Wesley Longman.

SMITH S. L. and MOSIER J. N. 1984. *Design Guidelines for User-System Interface Software*. Report ESD-TR-84-190, Mitre Corporation, Bedford, MA.

SUTCLIFFE, A. G. and MCDERMOTT, M. 1991. "Integrating methods of human-computer interface design with structured systems development," *International Journal of Man-Machine Studies*, **34**(5): 631–655.

THIMBLEBY, H. W. 1980. "Dialogue Determination," *International Journal of Man-Machine Studies*, **13**(3), 295–304.

THIMBLEBY, H. W. 1986. "Experiences with Literate Programming Using CWEB (A Variant of Knuth's WEB)," *Computer Journal*, **29**(3): 201–211.

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

THIMBLEBY, H. W. 1990b. *User Interface Design*, Addison-Wesley.

THIMBLEBY, H. W. 1991a. "Can Anyone Work the Video?" *New Scientist*, **129**(1757): 48–51.

THIMBLEBY, H. W. 1991b. "The Undomesticated Video Recorder," *Image Technology*, **72**(6): 214–216.

THIMBLEBY, H. W. 1993a. "Combining Systems and Manuals," *Proceedings BCS Conference on Human-Computer Interaction*, **VIII**, HCI'93, J. L. Alty, D. Diaper and S. Guest (eds), 479–488, Cambridge University Press

THIMBLEBY, H. W. 1993b. "The Frustrations of a Pushbutton World," *Encyclopedia Britannica Yearbook of Science and the Future*, 202–219, Encyclopedia Britannica.

THIMBLEBY, H. W. 1994. "Formulating Usability," *ACM SIGCHI Bulletin*, 26(2): 59–64.

THIMBLEBY, H. W. 1996. "A New Calculator and Why it is Necessary," *Computer Journal*, **38**(6): 418-433.

THIMBLEBY, H. W. 1997. "Design for a Fax," *Personal Technologies*, **1**(2): 101–117.

THIMBLEBY, H. W. 1998. "Design Aloud: A Designer-Centred Design (DCD) Method," *HCI Letters*, **1**(1): 45-50.

THIMBLEBY, H. W. 1999a. "Specification-led Design for Interface Simulation, Collecting Use-Data, Interactive Help, Writing Manuals, Analysis, Comparing Alternative Designs, etc," *Personal Technologies*, in press.

THIMBLEBY, H. W. 1999b. "Creating Discerning Users," submitted to Interact'99.*

THIMBLEBY, H. W. 1999c. "Hyperdoc: a Prototype Integrated Design Assistant," submitted to *Personal Technologies*.*

THIMBLEBY, H. W. and LADKIN, P. B. 1995. "A Proper Explanation When You Need One," in M. A. R. KIRBY, A. J. DIX and J. E. FINLAY, eds., *BCS Conference HCI95*, People and Computers, **X**: 107–118, Cambridge University Press.

THIMBLEBY, H. W. and LADKIN, P. B. 1997. "From Logic to Manuals Again," *IEE Proceedings Software Engineering*, **144**(3): 185–192.

THIMBLEBY, H. W. and WITTEN, I. H. 1993. "User Modelling as Machine Identification: New Methods for HCI," *Advances in Human-Computer Interaction*, H. R. Hartson & D. Hix, editors, **IV**: 58–86.

WOLFRAM, S. 1991. *Mathematica*, 2nd ed., Addison-Wesley.

* This reference will be replaced with a technical report if necessary.

* This reference will be replaced with a technical report if necessary.

YOUNG, R. M. 1981. “The Machine Inside the Machine: Users’ Models of Pocket Calculators,” *International Journal of Man-Machine Studies*, **15**: 51–85.