Solutioneering in user interface design

HAROLD THIMBLEBY with WILL THIMBLEBY
Stirling University, Scotland, FK9 4LA

Introduction

Computerisation is the technological imperative, yet it often results in deeper social malaise than the simpler methods replaced; the alienations, even inefficiencies, that they represent become more rigidly entrenched (Weizenbaum, 1979). All this is well known and widely debated. It is systemic and seems intractable (Illich, 1973). This article discusses a similar problem at the level of user interface design, a much smaller domain and one where there can be more hope. Here, there is usually just one responsible designer, or a small common-minded team, working with explicit utility functions (Ehn, 1988). The technology of interactive systems itself supports rational and experimental design (Thimbleby, 1988), and there are methods to improve designs, from practical (Gould & Lewis, 1985) to macro-theoretical (Moser, 1990) and micro-theoretical (Dix, 1991).

Designers work under heavy constraints, and part of the problem is that they have to solve problems from the current position. One might instead work backwards (Polya, 1957) from the overall desired effect of the system being designed, rather than forwards from the current already satisficed solution (Simon, 1969); one might also design more abstractly, specifically to use a formal approach to avoid implementation bias (Jones, 1980) — which would encourage searching for solutions in terms of the present implementation instead of the original requirements.

The user of most systems is unaware of the design history of those systems and the intermediate thinking of the designers: therefore solutions to *design* problems are rarely solutions to *user* problems. It is ironic that users often need to appreciate design history to make sense of the system they are currently using; designers call this ‘compatibility,’ and it is often compatibility with some product that was inefficient, inappropriate, and obsolete.

The aim of this paper is to encourage more considered design by discussing one of the consequences of narrow problem solving. We discuss a way in which designers solve their own problems, rather than address broader issues of user-centred design. We use the term ‘solutioneering’ for this. Having available a word for an attitude helps it to be consciously mastered.

Introductory examples

My son Will (aged 11) was inspired to write a HyperCard stack (Apple, 1987a) for a younger brother to practice arithmetic. He designed a ‘number maze,’ a maze of walls and locked gates. To open a gate in the maze and make progress, the user had to select the right answer, from amongst those offered, to a numerical question. For example, the first gate had an array of buttons, numbered 1 to 9. A question like, what is 4+3 is asked. To progress the user is supposed, in this case, to press the button 7. After three successful answers, the gate opens (figure 1), clearing the way forward to a lily pond and distant castle.

There was a bug, however. The script that generated random sums would occasionally generate sums totalling 10, more than 9. Will asked me to help him debug this part of his stack. I offered to help if he would print out or write down the offending part of the system, but he didn’t. When I asked later that evening how he had got on, Will replied that he had solved the problem. He had added a new button for 10 (figure 2).

Example two. My video recorder remote control has a clock, which can be set by pressing various buttons (Thimbleby, 1991). The way of pressing buttons is difficult to remember, and since clocks need adjusting for summer and winter time, this is a serious problem, particularly since users obtain no practice in the intervening interval. (For video recorders, unlike other clocks, one really has to change the time, rather than take the easier option of remembering that the time displayed is an hour out!) The manufacturers have obviously noted this problem, and have solved it. They have added a summer time adjust button.

Like my son, they were faced with the problem of either redesigning the existing system or adding an extra button to provide a solution to a problem that should not have arisen. It is easy to imagine user testing might have suggested a summer time adjust button, since it certainly is a solution to the apparent problem.

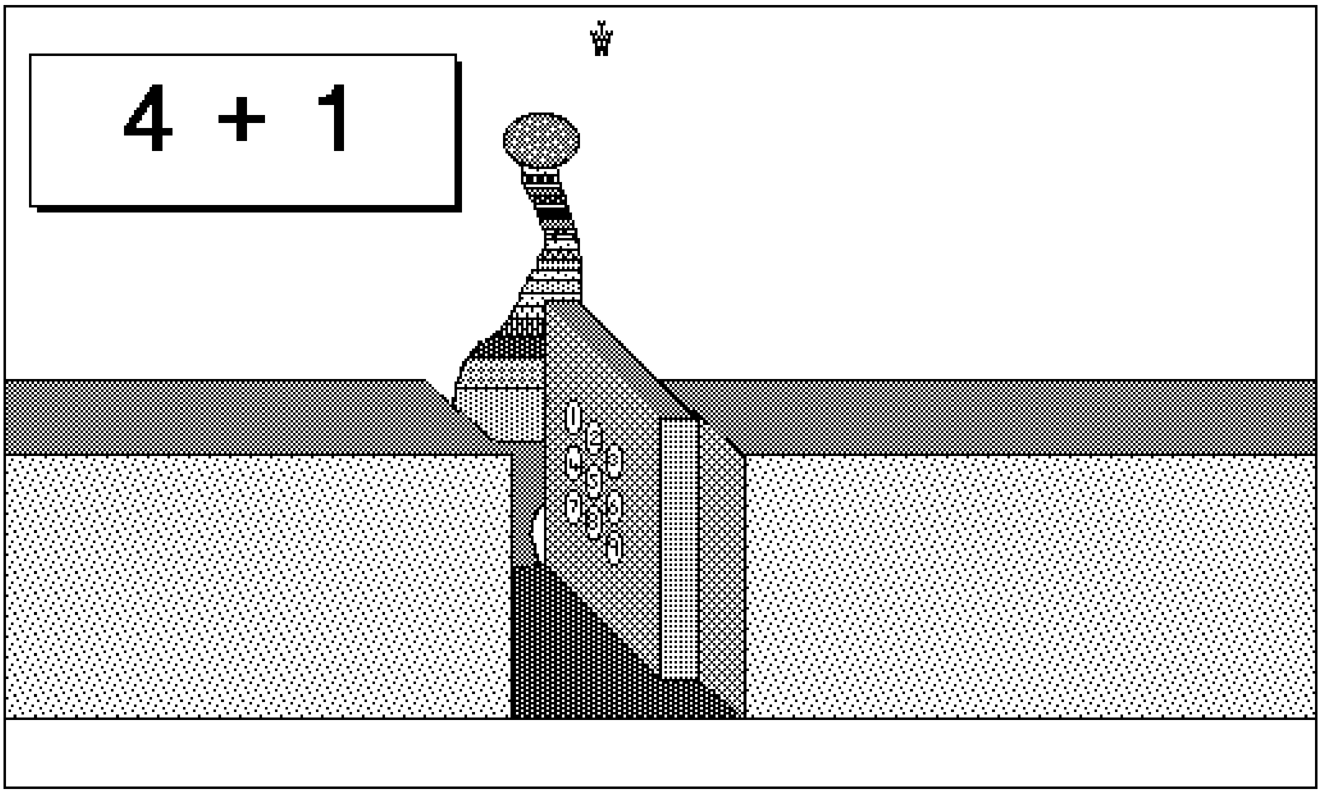

Figure 1.

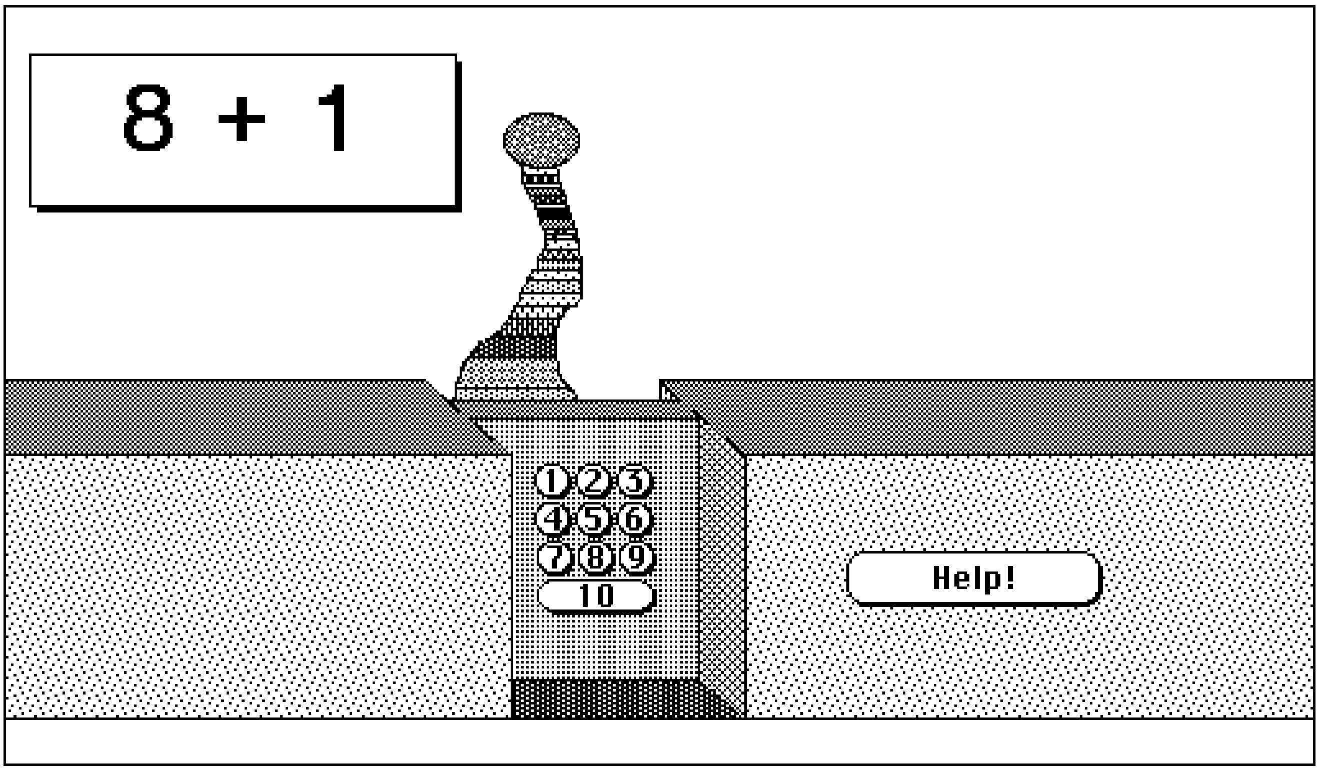

Figure 2.

Solutioneering

We call such an approach *solutioneering* (Thimbleby, 1990), and it is clear that solutioneering provides palliatives that are unlikely to reach to the root of the problem. Winograd and Flores (1986) argue that, far better than finding ‘solutions,’ is to avoid the problem — *dissolve* rather than *solve*. In this view they are supported by many authors such as Whitehead (1911) who claimed that civilisation advances, not by thinking (solving problems) but by finding ways to avoid thought (dissolving problems). One might have thought that the purpose of designing interactive systems, number mazes, video recorders, was no less: to make life easier for the user. (The number maze is intended to make life easier for the user *later* rather than at the time of interaction: the system does this by being hard to use, that is, educational.)

Solutioneering is easy to justify.

The so-called ‘democratic fallacy’ is that everyone should have facilities to suit their personal needs. In typography this results in manufacturers providing many fonts (Black & Stiff, 1992). The problem becomes not in having enough fonts, but in having enough users who know how to use the fonts that are already available. The fallacy is that manufacturers are not so much responding to demand as fostering it. With fonts, they are succeeding in creating a generation of type-fanciers and addicts who need more and more fonts to satisfy their self-feeding cravings. Like many addictions, fonts have the positive-feedback property that the more one knows, the more sophisticated distinctions can be made, hence the more fonts are required, and so one learns yet more. One also gets more exercise at solving problems in the preferred font, and hence that family becomes to seem easier than any other; one appears to become more proficient than eclectic typographers. Even at the level of the user, all this is solutioneering, since the purpose of fonts is to help text communicate with its readers, not to satisfy its author’s or designer’s prejudices.

An example of solutioneering in user interface design can be seen in almost all simple interactive systems, notably text editors. Initially a command language is designed, based on single letter mnemonics. Thus `Q` stands for quit, `C` stands for correct. But later the designer realises that we need a copy command, and chooses `Y` since `C` is already taken and `Y` is still free. Many systems use `X` for exit as `E` has been used for some other purpose. Surely a suitable point to reconsider the original commitment to single letter mnemonic commands, for the rationale is no longer valid. It is a solution of sorts. It is, however, a solution for the designer and not the user. Solutioneering of this sort is given credibility by user interface standards such as those of Microsoft's Windows (Microsoft, 1990), since menus can display `Execute`, `eXit`, `Correct`, `copY`. Alternative approaches are readily designed (e.g., Thimbleby, 1987) that avoid such arbitrariness and limitations.

Apple's System 6 provides a bugged method for copying files from one folder to another. When a file ('HCI chapter' in figure 3) is being copied, the copy can be 'Cancelled' by clicking the mouse on the Cancel button. However, if the file already exists in the target folder (directory), it will be deleted even if the operation is cancelled. Despite the Apple interface guidelines (Apple, 1987b), Cancel stops an operation under way, rather than cancelling its effect. (Evidently, copying is implemented as first deleting any same-named files in the destination.) System 7.0 provides a solution, by solutioneering, by changing the name of the button from Cancel to Stop (figure 4). This example is taken from (Thimbleby & Witten, 1992).

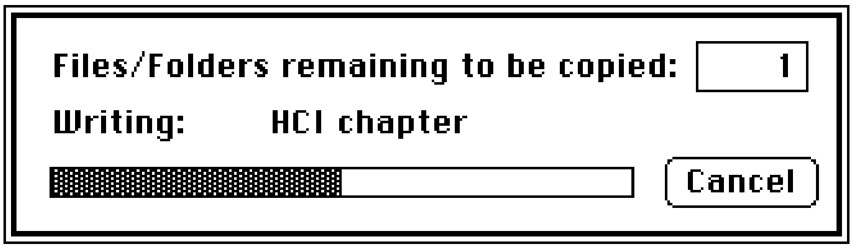
Files/Folders remaining to be copied: 1
Writing: HCI chapter
Cancel

Figure 3.

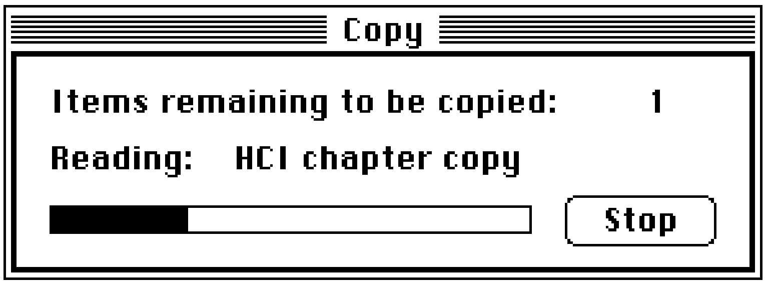
Copy
Items remaining to be copied: 1
Reading: HCI chapter copy
Stop

Figure 4.

The HyperCard stack Will developed was tested on its target user, Isaac. User evaluation showed that the task needed modifying, and should be harder. It was, therefore, quite appropriate to add extra buttons so that more complex sums could be set. In this light, the original bug might be seen as a serendipitous event to improve the system (cf. Roberts, 1989) — though retaining the bug resulted in a system harder to use than originally intended! One wonders if the video recorder's summer time adjust button comes into this category: did the designers of the remote control consider their solution with respect to the intended task? The provision of the summer time adjust button actually makes the clock harder to use, for several reasons. First, it reduces the frequency that the user adjusts the clock in the normal way. Therefore when the clock does need adjusting (e.g., because of battery replacement), the user is unpractised. Secondly, a single button is so easy to press that it can be done by accident, and indeed often is, with the result that the user is now often readjusting the clock. Surprisingly there is no button indicated for winter time adjust; instead, the summer time adjust button has to be held down for two seconds. One only remembers this simply because it has to be used every time the remote control is picked up from under a pile of newspapers that have inadvertently advanced the clock by summer time adjustment.

The documented bug

When changes to the user interface cannot be made (for example, the software has already been burnt into ROM), the solutioneering approach suggests modifying the manual, Brook's (1975) 'documented bug' approach. We avoid giving examples here (cf Thimbleby, 1990), but raise the point that a documented bug creates the impression that the bug is an essential feature of the design. Having documented a bug, attention can be diverted away from the design to helping the user cope with the resulting problems. Problems with the design become problems of use, and hence can be conveniently characterised as problems of psychology rather than of design or engineering.

Nader (1965) gives the example of a faulty automobile parking brake: the design question, as framed by the industry itself, was not correcting the poor engineering, but asking how to train users (drivers) to read manuals and how to write those manuals better! From user interface design is the psychological problem of ‘getting lost in hyperspace’ (Thimbleby, in press). Rather than ask how to redesign a system so that navigation is easier (even feasible, i.e., formally computable), the solutioneering approach is to ask, “*given* that users get lost, how can we reduce the consequences?” Finding answers to this simpler conditional problem frees the designer to make more complex networks than before.

Summary

Interactive systems, particularly domestic consumer products, grow buttons and other features to solve problems caused, in a large part, by bad design. Buttons have the advantage — for the lazy designer — that they can be added to systems without reconsidering the original design or the task the system as a whole is really meant to support. Surely designers should be solving the user’s problems, not finding ways to solve their own problems. ‘Solutioneering’ is a good word for describing this tendency to solve the wrong problems, perhaps even impressively. A sad consequence of solutioneering is that, since designers *have* solved problems, users continue to be blamed for ‘their’ problems.

References

APPLE COMPUTER INC. 1987a, *HyperCard User’s Guide*.

APPLE, INC. 1987b, *Human Interface Guidelines: The Apple Desktop Interface,* Addison-Wesley.

BLACK, A. & STIFF, P. Oct. 1991, British Computer Society Publishing Group Meeting, London. Cited in MORGAN, C. 1992, “Typefaces Meeting,” *TEXline*, no. 14, pp18–20.

BROOKS, F. P. 1975, *The Mythical Man-Month*, Addison-Wesley.

DIX, A. J. 1991, *Formal Methods for Interactive Systems*, Academic Press.

EHN, P. 1988, *Work-Oriented Design of Computer Artifacts*, Arbetslivscentrum, Stockholm.

GOULD, J. D. & LEWIS, C. H. 1983, “Designing for Usability—Key Principles and What Designers Think,” *Communications of the ACM*, **28**, pp300–311.

ILLICH, I. 1973, *Tools For Conviviality*, Calder & Boyars.

JONES, C. B. 1980, *Software Development: A Rigorous Approach*, Prentice-Hall.

MICROSOFT CORP. 1990, *Microsoft Windows 3.0*, Document SY06851–0290.

MOSER, P. K. ed. 1990, *Rationality in Action*, Cambridge University Press.

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

POLYA, G. 1957, *How to Solve It*, 2nd ed. Princeton University Press.

ROBERTS, R. M. 1989, *Serendipity*, John Wiley & Sons.

SIMON, H. A. 1969, *The Sciences of the Artificial*, MIT Press.

THIMBLEBY, H. W. 1987, “A Menu Selection Algorithm,” *Behaviour and Information Technology*, **6**(1), pp89–94.

THIMBLEBY, H. W. 1988, “Delaying Commitment,” *IEEE Software*, **5**(3), pp78–86.

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

THIMBLEBY, H. W. 1991, “The Undomesticated Video Recorder,” *Image Technology*, **72**(6), pp214–216.

THIMBLEBY, H. W. in press, “Heuristic Strategies for HyperText,” in NATO ASI Series F, Proceedings NATO Advanced Research Workshop on Mindtools and Cognitive Modelling, *Mindtools: Cognitive Strategies for Modeling Knowledge*, KOMMERS, P. A. M., JONASSEN, D. H. & MAYES, T. eds.

THIMBLEBY, H. W. & WITTEN, I. H. 1992, “User Modeling as Machine Identification: New Design Methods for HCI,” in *Advances in HCI IV*, edited by HARTSON, R. & HIX, D., Ablex,

WHITEHEAD, A. N. 1911, *An Introduction to Mathematics*, William & Norgate.

WEIZENBAUM, J. 1979, “Once More: The Computer Revolution,” in *The Computer Age: A Twenty-Year View*, DERTOUZOUS, M. L. & MOSES, J., MIT Press.

WINOGRAD, T. & FLORES, F. 1986, *Understanding Computers and Cognition*, Ablex.