Computer Literacy and Usability Standards [*]

Harold Thimbleby
Stirling University, STIRLING, Scotland, FK9 4LA.
Email: `harold@uk.ac.stirling.compsci`

There is a tension between making computer systems more standardised (and hence easier to use given previous experience with other systems) and whether users should be expected to be more competent, to be computer literate. Computer literacy enables users to cope with non-standard, *ad hoc*, or badly designed systems; instead it should make users campaign for standardised systems that are easier to use.

I. Introduction

Standards are a way of reducing the cost of transferring skill from one system to another. Reducing costs is a benefit at all levels: for users of systems it means that their skills are more marketable, and that they will have reduced training times in new jobs; for both user and employer, standards reduce errors, because the skills of the user transfer and because appropriate commands for one system are equally appropriate, certainly not erroneous, on another.

At one extreme, **lack** of standardisation has the advantage that systems can be tailored exactly for their specific environments; at the same time specificity increases staff retention—users (following training) are less mobile than those with training on standard systems. Yet non-standard tailored systems are not inevitably beneficial: few software companies are technically able to produce quality solutions, and unforeseen difficulties with one-off systems may outweigh anticipated, known, problems with off-the-shelf solutions. Lack of standards implies a lack of external guidance in the design process. At the other extreme, **strict** standardisation stifles manufacturers' creativity, but it does ensure that end user skills are more readily transferable. One must decide whether it is better to retain eventually demoralised staff or to be able to more readily recruit staff with higher initial skills.

Rather than emphasising standardisation of systems, we could train users in the underlying concepts. Users would know that discs have to be formatted however a particular computer system did it. It takes training to know how use a computer, to control a word processor or to format a disc. As more businesses adopt computers for their everyday running, so there is an increasing demand for trained people to work effectively in the computerised environment. It has reached the point where people cannot function at work unless they are computer literate.

II. Literacy

Computers are only one of many modern gadgets, and why should they be unique in being difficult to use? We don't need, and don't expect "telephone literacy" courses—yet telephone exchanges and satellite links, and so forth, are terribly complex.

The point is that you can easily use complicated things when they are properly designed for you. Or again, we may have car driving lessons, but it is only the enthusiast or professional who has or even wants "car literacy" (engineering) lessons. The rest of us find it quite sufficient to learn how to drive cars, we do not need, and should not need, to have to learn how to fix their mechanical problems.

[*] *User Needs in Information Technology Standards*, pp223–230, C. D. Evans, B. L. Meek and R. S. Walker, general editors, Butterworth-Heinemann, ISBN 0–7506–1559–1, 1993.

There was a time when the Government was promoting computers in schools. This was like the Victorians having a national initiative to buy every school a model railway set. You can imagine someone saying that children, if they were to get on in the world, should know how railways work. Children would learn all about railways from using the school layout.

The Victorians never did “railroad” schools, and the idea seems faintly absurd. Real railways are very different from model railways—British Rail or the French SNCF are mostly in the news because of their safety record, and safety is not an issue that much could be learnt about from a toy layout. So, toy railways may appeal to little technologists, but they don’t make people “railway literate,” at least not in a way that would be much use to an adult railway employee. Likewise, school computers are not helping anyone to be computer literate in a useful way. Adult computer literacy is also suspect.

Real computing (let’s say as would be used for the control of a nuclear reactor) is very different from anything that can be taught at school. To give some idea of the gulf between school computing and real computing, consider another simple analogy: Suppose we have a child here playing with Lego bricks. She can make houses, she can make bridges, she can make cars. We are not led into thinking that she will grow up to be a competent builder, bridge designer, car designer—because these things require much more thorough design than can possibly be explored in Lego. You can’t just make a bridge over a river by pressing plastic blocks together; it wouldn’t be safe and wouldn’t even support its own weight. Instead, we see the child simply at useful play for its own sake rather than pompously imagining the child is on a “building literacy” course!

This point is not so clear with computers. The trap is that real computers look rather like toy computers in a way that real bridges cannot look like Lego bridges. Unfortunately I cannot easily tell by looking at my computer whether the programs it is running were designed by a three year old or a professional programmer. But you can tell by looking at the Forth Road Bridge that its designer knew something of bridge building.

There is no way you are going to think little Jimmy here is going to grow up to be a bridge builder just because he can make a ten inch Lego span! But we do make this mistake with computers. We exaggerate the importance and effectiveness of so-called computer literacy. “If Jimmy can write a ten line program in BASIC, why, he can get a job in pay-rolls!” Total nonsense.

Computer literacy, when it starts at school, is misplaced. Instead, think of the computer as a toy, like Lego, something that children can develop many and various skills with—the least important of which is computer literacy itself. Lego teaches manual dexterity, solid geometry, problem solving, mechanics; computers teach touch typing, drawing, music composition, writing, problem solving, even chemistry and French.

III. Who is responsible?

Ralph Nader’s classic book *Unsafe at Any Speed* woke up the 1960s car industry: cars must be designed for safety.

Nader strongly criticised the car industry for making intrinsically unsafe cars and blaming the driver for accidents. “The driver has the accident and the driver is responsible,” the manufacturers argued. The car had the accident too, very often caused by its poor suspension or feeble brakes. Pedestrians who were ‘gently knocked’ were killed by being slit from throat to stomach by being cut with sharp body styling. Car manufacturers argued that in any collection of accident statistics one would be bound to get some gruesome cases: they denied that the inherent dangers of the styling were their fault, and anyway drivers wanted such grotesque styling! Nader’s campaigning eventually made car designers take responsibility for designing cars properly. He gave them a conscience for safety which soon became enshrined in standards and safety legislation.

In the sixties people had problems with cars, whereas today people have problems with computers. In the sixties people were told to drive more safely (to be “car literate”). Today people have problems with computers. Today people are told to read the manual and become computer literate. When a user does that the problem goes away, or the problem becomes one of writing understandable manuals. It is easy, but mistaken, to think usability is a problem of educating users rather than manufacturers. (That’s where standards comes in: education for the manufacturers.)

With cars the real issue was not getting better driving nor making drivers “car literate.” That would have been impossible since many designs were intrinsically unsound and could not even be driven well by highly skilled drivers. The onus—not admitted in the sixties—was actually on the car designers to make cars that were easy and safe to drive, and that in turn required standards: legislative and professional. Likewise the onus for better computer use is very much on the designers, not the users. When users, instead, are encouraged to become computer literate, manufacturers are making users' supposed lack of skill a scape goat for their bad designs.

Today, when you hear that children can use video recorders, you get the strong feeling that you are old and past it—having missed out on suitable training to deal with this sort of complicated technology. That is exactly what the manufacturers want you to think! They want you to think that the difficulty you have with their systems is your fault, not theirs. If you have heard that children can work it, then doesn't that just confirm that it is all your fault?[1] Users are made to feel untrained, and if they feel that way then the solution would at first sight be to get trained, to become computer literate. Users should not be feeling that way in the first place!

IV. The relevance of user skill

General computer skill is not important, as will be argued by a brief digression.

I once saw an old, green leather-bound pair of volumes on engines (French, 1908). Written shortly before the First World War, its author enthused about engines, or "modern power generators" as he called them. There were all sorts of engines: steam engines, gas engines, oil engines, and petrol engines. (It is interesting that this particular author saw no future in petrol engines: they were unreliable, needed hard-to-obtain fuel, needed gears—whereas steam engines were reliable, used water and easily obtained coal or wood, and didn't need gears at all. Well, nobody can foresee the future!) The author's main point was: to get on in the world—the Edwardian world of engines—one really needed to *know* about engines. His two volumes told the reader everything there was to know about engines.

Today we have more engines around than this man could possibly have imagined! I have an electric engine on my wrist—a watch. I have four in my washing machine; and there are eighteen in our kitchen at this very moment (not counting the engine in the toy electric car that is in bits on the table).

In the old days, when this engine book was written, if a farm had only one oil engine it was important that the farmer understood it. If it went wrong, he would have to know something about engines in order to fix it. But today a farm has thousands of engines, and if one goes wrong (say, the starter motor in the farmer's car) there are many other engines to get by with. There are so many engines and they are so well hidden that the farmer no longer thinks of engines as such, but of cars, tractors, clocks, sprays. Engines have disappeared; instead there are appliances that do useful things. So, too, has the need for engine literacy disappeared. What has happened is that "engines," so far as their users are concerned, have become standardised. An engine user, once called an engineer, no longer has to treat each sort of engine differently. Engines are simply started with the flick of a switch. It is important to emphasise that the technical diversity of engines, that once required a corresponding breadth in user skill, has now disappeared from the user's point of view.

Likewise, computers are beginning to disappear inside the things that they make work.

You can now get a toaster with a microchip in it. The toaster is a toaster and you don't need to be computer literate to use it! Our washing machine has one too, as has our microwave, the central heating, the car, the TV and video recorder. To have to be computer literate to use a toaster or car is ridiculous! Instead, the toaster *is* toast-literate. The computer in it knows about toast (I hope) so you don't have to know about it. Only when the toaster does silly things (does it crash?) do you need to be computer literate in any sense—and, in this case, if you *do* have any sense you'd take it back to the shop and get one that worked properly! In terms of standards for user interfaces, the toaster conforms to toaster user interface standards.

The thrust of computer literacy, as these examples indicate, is misplaced. It is only because current computer systems are so bad that we need training to use them. Better to make the systems easier to use; far better to make the computers–as–such disappear and the users' view of them conform with what tasks they are doing. The problem runs very deep, and is not so easily solved as ignored.

V. Standards for "people literacy"

Computer literacy—people being trained to be computer literate—is a diversion. We need "people literacy"—that is, computers designed to be human literate so that the people who use them can simply get on with their job. That is why standardisation is needed.

[1]A naïve child is certainly at an advantage over an adult who thinks he knows what he is doing but is wrong!

The reality, though, is that people do want jobs now, working in today's computerised environments and, sadly, for this they need to be computer trained to work effectively. Computer literacy, then, is a training for the real, rather than the ideal world. Let's not forget, however, that it is a stop-gap, and in the long run we would be better off getting the computers up to human standards rather than reducing the humans to cope with arcane computer etiquette.

Users require specific training and skills with each system that they encounter. This increases job satisfaction momentarily, but in the long run it severely reduces mobility. If you can only get a new job with experience of A, if you have spent months learning B (or even A'), you will be stuck. The better idea, still restricted to research systems, would be to make each system *customisable* and *adaptive* to the particular user working with it.

VI. Standards & quality versus protectionism

Manufacturers resist standardisation. It restricts their freedom to implement in ways that suit them, in cheaper, more impressive, or technically easier ways. Designers—not being encumbered by standards—can add and remove features at will, for secondary purposes. Systems can be implemented by cheaper, unskilled programmers. Without standards to judge interactive systems for quality, it follows that designers can choose to increase market penetration deliberately (or innocently) by compromising user interface quality. Indeed, there is no requirement that individual components of a complex system have to be consistent even with each other, adhering to internal standards. Importantly, standardisation opposes the current trend for legal protection of software systems' *proprietary* appearance.

The current law for protection of design makes it very hard to protect conceptual ideas, and computer programs in particular. Therefore manufacturers protect their development investment by protecting what can be protected: the so-called 'look-and-feel' of their products. If another company releases a product with a similar user interface, that company is potentially infringing look-and-feel. Protecting normal development costs in this way works against producing standardised user interfaces; sadly most of the modern (windows, mice, menus) style user interfaces are very much bound up with look-and-feel. Stallman and Garfinkel (1990) are leading a stand against this essentially restrictive practice.

A major problem for the entire industry is that protectionism stifles improvements: even if I know a better way of doing what some manufacturer has provided, I cannot start with an improvement on their product. From a technical point of view most user interfaces could very easily be improved (you have only to look at the so-called macro languages of commercial spreadsheets to see that computer science seems to have passed them by). There is no opportunity to improve commercial systems because of copyright and look-and-feel restrictions. Evidently manufacturers don't want to improve their own products, and they don't want anybody else to do any better either.

In consequence, for the consumer market, lack of standardisation between, as here, word processors results in misplaced customer allegiance: users become reluctant to invest in alternative manufacturer's software. Ultimately people start 'standardising' on the very *worst* systems because these are the systems that least prepare people for anything else.

If I could legally reverse engineer a system, I could build a new version of it with fewer bugs. I could design my program carefully rather than let it accrete *ad hoc* code over years! I could use fewer, more reliable programmers. I could give my program a decent warranty (which no commercial software has so far aspired to).

Like the example of reverse engineering to improve a system, both engineering and scientific progress *depend* on copying to improve existing designs with known performance. (The conventional paradigm is that experiments are published, claims made, and others then check the results: by repeating and trying variations.) When this is illegal nobody can systematically improve the many interrelated, design-specific features such as ease-of-use. If nobody can check up or attempt to improve designs, there will be a powerful temptation to implement systems laxly. This fosters the impression that user interface design is easy! In short, without standards for user interfaces, manufacturers become secretive, their system implementation quality decreases ... which in turn encourages them to be increasingly secretive.

The only way to enable anyone to improve systems is to standardise on them. Standardisation would remove the threat of litigation over copying user interface designs. Instead, manufacturers would have to compete by making their systems better. To use an analogy: nobody doubts that Ford and Rolls Royce copy the basic look-and-feel that is a car. Ford and Rolls Royce compete not on how steering wheels are used, but on how their cars handle, how they accelerate, how comfortable they are. Cars are a more definitive concept that word processors or operating systems. Try and imagine what makes a general car (a list of wheels, seats, engines, down to fuel gauges and speedometers on the fascia). The corresponding exercise even for word processors doesn't get off the ground.

It is time we started developing "standardised"—that is, deliberately public—user interfaces. It is now thirty years since Algol 60 ... where is the effort to make a similar impact in user interfaces? Imagine word processor manufacturers vying to produce ISO Level 4 word processors cheaper, faster, more reliable, easier. It would generate the sort of competition that would be of great benefit to users. (This argument is developed further in Thimbleby, 1990.)

Standardisation *is* a realistic goal for user interfaces, if not always easy. We need only look at the user interfaces provided for programmers—programming language standards—to see that standardisation is perfectly possible. Programming languages like Cobol or Ada are *fundamentally* more complex than any user interface in widespread use. User interfaces may be *superficially* more complex, but this appearance is a result of arbitrary, inconsistent, interaction techniques provided as incoherent, piecemeal solutions to jobs simpler than programming. Incoherence makes user interfaces complex by quantity rather than quality.

Programming languages became standardised by two main routes. First, most (but not all) language designers are academics with no financial stake to make a market niche, rather they intend to design a first class language; second, standards-enforcing bodies (such as the US Department of Defense that initiated Ada) recognise the advantages for programming language standards. Whatever the pitfalls and commercial subterfuges, the result of standardisation is that programming skill improves, and programmers' retraining costs are reduced (programmers need less retraining the more a standard language like Ada is required). One might ask why users don't have such as spokesman as the DoD.

Until there is an undertaking to standardise user interfaces, user interface design and usability will be a stagnant area. All work in usability today is about psychological problems, user modelling, explanation, training: none of this is making any system any better!

Without standards, user interfaces are an after-thought. If one can get away with any user interface, anything goes. Thus it is easy! In contrast, programming language design is recognised to be hard. That is why there have been many highly motivated and successful moves to standardise languages, and hence share development efforts, improve quality, and so forth. (It is cynical to point out that programmers have a vested interest in standardised languages for advancing their own job prospects.)

If a programming language has a design or implementation error in it, everything done with it is at risk. But as computers don't tolerate errors, errors such as dividing by zero are very hard to conceal or ignore. Programming languages and their implementations work against an intolerant judge: whether they will work or not on a computer. User interfaces, on the other hand, work against a tolerant and powerless judge: the user. Design errors comparable in severity with dividing by zero, like irretrievably deleting a day's work "by mistake," are rarely perceived as the inevitable consequence of fundamental design faults.

User interfaces won't be standardised while users accept badly designed systems. While users accept badly designed systems, designers can get away without putting in any effort into new designs. Designers will continue to believe standardisation is irrelevant—and users will continue to seek support in computer literacy courses.

VII. The proper use of literacy to advance standards

In summary, computer literate users should not understand how to cope with badly made, one-off and non-standard systems. They should use their literacy to recognise poor quality for what it is, embark on litigation, complaint, and generally be as intolerant of bad design as the computers they now understand.

Acknowledgements

This chapter owes much to discussions with Greg James (1991) and Bruce Macdonald.

References

James Weir French, *Modern Power Generators* (two volumes), Gresham Publishing Company, 1908.

Greg James, "Computer Illiteracy Just an Excuse for Bad Interfaces," *Gauntlet*, June 20, p2, 1991.

Ralph Nader, *Unsafe at Any Speed*, Pocket Books (Simon & Schuster): NY, 1966.

Richard Stallman & S. Garfinkel, “Against User Interface Copyright,” *Communications of the ACM*, November 1990, pp15–18.

Harold Thimbleby, *User Interface Design*, Addison-Wesley, 1990.