

Formulating usability

Harold Thimbleby
Stirling University, Stirling, Scotland, FK9 4LA
Tel: +44 786 467421
FAX: +44 786 464551
EMail: HWT@CS.STIR.AC.UK

October 22, 1998

Abstract

Usability is empirical, and often highly context-specific, but it would be useful for designers to have general estimates of usability from interactive system specifications alone. We discuss how this problem may be approached, and we give examples. We also discuss the justification for the approach, since it is unusual to measure usability without involving users. The explicit mathematical content of this paper has been deliberately kept to a minimum.

1 Introduction

User interfaces are getting more and more features, but there are very few systematic ways to make them any better, or even to know how good or bad they are. It is almost impossible to compare designs: first, because usability measures may not relate to identifiable design features; second, because one generally wants to compare two or more designs to spare the effort of implementing all of them — or the embarrassment of implementing the ‘wrong’ one!

The motivation of the present work is to make a contribution to user interface design, specifically to discuss ways of measuring usability from design, from specifications rather than implementations. The aim is to show that solid results can be obtained with very simple and general assumptions, and to illustrate certain usability criteria that can be established in the design cycle. Ideally, of course, user interfaces should be empirically evaluated and then improved, but in practice many products are designed and then fobbed off to users, with little opportunity for improvements. It is therefore crucial to have *design* support for usability measurement.

A tool supporting the sort of analysis promoted here is discussed in a companion paper with a more practical emphasis [13]; related work includes [11, 14].

1.1 Usability: empirical or platonic?

The quality of a system depends on it meeting various criteria, which are traditionally vague and non-constructive in design: according to an influential definition due to Brian Shackel usability is “the degree” to which specified users can achieve specified goals in specified environments, subject to various comfort, effectiveness, efficiency, and acceptability criteria [8]. Shackel’s influential definition suggests terms should be given numerical values; designing for usability has recently become called ‘usability engineering,’ almost lending an ‘engineering’ status to any such numbers! But suppose only 64% of users can use a system “acceptably,” when 70% was the criterion specified . . . How can the system be modified to increase acceptance? Does the difference matter? How was 70% justified anyway? How can acceptance be predicted in design stages?

Shackel-type usability criteria are certainly relevant for users entering contractual arrangements with suppliers; however, such operational definitions are not constructive for designers. Without theories it is impossible to use empirical measures systematically to improve system design. What has been called usability engineering, therefore, ought to be called (say) empirical usability. Empirical usability will be seen to be complementary to the approach taken here. which might be called, in contrast, formal usability. The activity of assessing formal usability is called formulating usability. (This approach isn’t

engineering either! It is, however, more readily involved in the engineering aspects of the design process than empirical usability.)

Most formally motivated work in user interface design is based on considering predicates, addressing such issues as whether ‘undo’ undoes everything [4, 7, 10]. Though this approach is extremely useful (and has proven fruitful in research terms), a design either satisfies such properties or does not (or the property is intractable). Thus, whether a design is more usable than another comes down to the question of whether the property concerned is appropriate or not. Many ‘obviously’ necessary formal criteria — such as, a system should be strongly connected — are not sufficient for usability (almost all unusable systems are strongly connected). Conversely, the necessity or desirability of a property *itself* may be subject to debate, particularly when the property is one of many that are arguably equally appropriate.

Finally, properties that are tractable are often so strong that they impose unreasonable constraints on the design; they may conflict with other properties. One then has to choose amongst the properties.

Thus, whereas formal properties might be supposed to establish usability, one ends up attempting to measure the usability implications of the properties! Another approach is to deem properties to be design requirements: this design-motivated approach, of course, influences the choice and nature of the appropriate properties [6].

1.2 Complexity

Usability is conventionally user and task-specific; it is also empirically established, almost by definition (Shackel, op. cit.). However, a designer knows from task analysis (and perhaps market research) only the set of appropriate tasks (and perhaps their frequencies) that should be supported in a product. There may only be the vaguest ideas about the population of users. In a commercial setting, the choice of features selects the users: users who don’t want something won’t buy it. The designer is therefore interested in measures of usability taken over all functions to be implemented regardless — so far as possible — of the actual or eventual users.

In computer science the complexity of algorithms are routinely compared, ignoring details of what sort of computer the algorithms are run on, though of course it is interesting that special purpose computers can excel in special cases. Likewise, we want to obtain measures of usability that enable design comparisons, without making assumptions about users, though, of course, we acknowledge that certain users with special skills or knowledge may be able to do much better, or even do much worse.

The complexity of algorithms is often (though not inevitably) couched in such terms as to make it independent of the actual speed characteristics of computers. This has considerable advantages when comparing algorithms as such, as opposed to comparing computers, or their performance on particular problem cases.

Indeed, having a general measure of usability will help designers have a better picture of the type of users who will be advantaged or disadvantaged by the user interface, or how to modify the design to make it more (or less) appropriate to those types of user. This is analogous to being able to compare Quick Sort to Insert Sort. Under different assumptions one method is better than the other; further, under different assumptions, one would sensibly optimise the algorithms in different ways. As in programming, we shall also see that attempting to determine complexity measures often more quickly suggests ways of improving the design; in usability it is much easier to lose sight of the original issues, and get carried away with the improvements.

For example, an analysis of Insert Sort changes when you decide, as a result of initial analysis, that heaps, rather than arrays are a more sensible data structure; in user interfaces, you may decide, as a result of initial analysis, that usability would be much improved by providing help. Just as the way one implements heaps is crucial, the way help is provided is crucial — and may even be much more interesting than the rest of the design problem! However, in neither case, should the interest in the new design problem detract attention from the original goal of improving efficiency or usability. Terrific heaps, or wonderful help, are not the point if the final system still lacks the efficiency or usability that was intended.

Just as practical programming analysis can consider the time taken to write a correct program as well as to run it, in user interface analysis, we can weight the time taken to read the manual, or otherwise learn to use the system. (Interestingly, some of the usability measures we will consider below correspond to lengths of certain sorts of user manual.)

1.3 Realism and accessibility

Much work in user interface design strives for realism. Realism shows itself most visibly in multimedia user interfaces, but it is also a trend in cognitive modelling. Realism, though valid for some purposes, makes design analysis harder. That is, as designers we want to analyse and understand as much as possible, without having first to make design assumptions, say, either about output devices or brains. Furthermore, when a ‘realistic’ analysis suggests that some design is harder than another, it is difficult to trace the reason for this back to specific design choices. In strict terms, it is very difficult to correctly identify fudge factors in complex usability models: therefore we are motivated in this paper to examine very simple models.

In the analysis of algorithms, realism makes life extremely and obviously difficult. Unfortunately, in user interface evaluation, realism rather confuses the issues. For example, it becomes very hard to generalise empirical results. Users may be seduced by (I mean, *like*) realism — in itself no bad thing — but this effect may make identifying ‘raw’ usability problems considerably harder. On the other hand, realism often seems to be used to conceal design faults that cannot be ameliorated in any other way.

In every branch of HCI, whether oriented towards computing or towards psychology, the emphasis is on coping with realism, that is, potential complexity. Thus programming systems are made as powerful as possible; psychological models cover as broad a domain as possible. The result is that researchers and designers may have convenient access to neither resource. Much as we may covet many modern powerful user interface development systems or cognitive simulations, they do not make a contribution to design unless one has access to them.

However a system design is to be analysed, there will be an immense benefit to be had of importing a body of well known theoretical concepts and measures. Representing an interactive system in an existing framework may appear limiting, but this ‘limitation’ guarantees any results, practical or theoretical, are not tied to an idiosyncratic framework or implementation, and it guarantees results can be widely understood, tested and applied. They might also be more reliable!

1.4 Emphasis on human or machine?

Lately, it has been emphasised that much work in human-computer interaction is technology-centred. There is (to my mind) a rather excessive reaction to making human-computer interaction human-centred, making things pleasant — but not necessarily correct. Of course, both aspects are essential: people must ‘like’ interactive systems, and they must *stay* liking them, which relies on the systems being technically competent and properly designed, and evaluated in the way being advocated in this paper.

At first sight the suggestions for formulating usability are asymmetric, concentrating exclusively on the computer system specification, to the exclusion of any assumptions about the human user. The methodology, however, is equally at home assessing the human’s usability from the computer’s point of view. It just happens that we are addressing the problem of design, rather than the (equally interesting) problem of educational programmes. It just happens that we are designing for people to use computers, not the other way around (although it happens); the design approach one should take in domains such as knowledge elicitation will be difficult to relate to our approach to formulating usability for exactly this reason.

In summary, an apparent emphasis on technology arises here, not because we rate technology higher than users, but because we want to better control the technology to be more appropriate for human users.

1.5 A choice of theory base

This paper suggests treating user interfaces as finite state machines, and using graph theory as an analytic framework. The approach is not contentious, but means that many (even obvious) faults in user interfaces will not be addressed.

In fact, formulating usability does not depend on this particular choice of theory base, but graph theory has the very considerable merit that it is well understood, feasible and sufficient to make the point. Graph theory has a huge and mature literature (going back to Euler in 1736), so many functions and properties are thoroughly understood — one can almost read any book on graph theory and take one’s pick of ideas to apply to usability! There are many computer packages available that make the practical evaluation of usability (in graph theoretic terms) essentially trivial [9].

Together with graph theory, we make assumptions based in information theory. (How, will be explained below.) Although information theory is somewhat ‘out of favour’ in psychology, we are *not* applying it to cognitive function. We are applying it to communication *between* human and computer, and we want to be able to formulate times (typically, minimum times) for effective communication.

However, we will ignore timeouts (where the system makes state transitions autonomously, typically, when the user does not press any button for several seconds). Almost all such concrete features that might have been introduced have questionable value in improving user interfaces *in general*.

Some other usability problems, however, can be addressed. A VCR’s remote control buttons are typically differently arranged and have different meanings (even though with the same name!) than the front panel of the VCR. This sort of difference becomes very apparent.

Buchner and Funke [1] have done some relevant empirical work on the usability of finite state machines.

2 Various complexity measures

What is the user trying to do? What does the user know?

Depending on the purpose indicated, we suggest the following approaches to user interface complexity analysis. Each example below shows that, for a reasonably formulated issue, we can obtain a useful measure of usability. If one knows about the particular circumstances of use, weights can be applied, for example, to allow for anticipated task frequency distributions.

When we say (informally), “how long does it take,” we mean: assuming a button press (and the system response) takes unit time; reading a paragraph in a manual takes unit time; and so on. We are obviously interested in complexity measures that are not biased by selecting particularly proficient or particularly slovenly users as benchmarks.

So, if we assume messages between computer and human are encoded as button presses and on/off lights, then we can easily establish how many presses or flashes is sufficient to communicate certain facts (such as, what state the system is in). More generally, the user interface will have all sorts of features that complicate the analysis! It may be direct manipulation. It may use icons and graphics. In which case, information theory comes to our rescue: however it is done, at least so-many bits must be transferred. The button counts (or other simplistic measures) are then simply concrete ways of counting bits. (If you can design the user interface so that the user has a ‘good way’ of ‘counting,’ then the numbers may be reduced logarithmically, or even better. For example, a 2-axis rotating 2D projection of a solid happens to be an *extraordinarily* efficient (and subjectively satisfying) way of conveying approximate details of the solid’s geometry). Clearly (and this is a standard observation), the number of bits required depends on the models assumed; this is where the user model comes in explicitly. The trick, then, will be to make as clear and as weak assumptions about the user model (in the precise information theoretic sense) as possible, or else we will be making profound and likely unsupported psychological assumptions.

More generally, all of the complexity arguments here are established on the basis that the computer is not helping (because we assume the purpose of the computer is to support the user’s tasks, which are not always to evaluate the system design). That is, although there may be a direct way to give the user a model, doing so can be assumed to interfere with the user’s task. Indeed, we are trying to establish the usability of a given system, rather than the usability of the system with optimal (possibly hypothetical) help, which is a *different* system and with different usability! In many cases, a modest amount of help improves usability (e.g., by eliminating the user’s memory demands). Thus formulating usability may, as a useful side-effect, indicate that providing help would have a significant impact on the design’s usability. Thimbleby [12], for example, gives some example *computer* algorithms that can improve usability to reduce the user’s problem of getting lost in the system graph; further examples are given below.

In most cases below we obtain *lower* bounds; it would be somewhat surprising to obtain upper bounds or even averages, as this could imply some performance obligation on the part of users!

Some of the most robust empirical results in usability relate to manuals and user training [5]. One observation, due to Carroll [2], implies that minimising a length complexity of manuals improves their utility, “less can be more” [3]. Thus, we may be safe in assuming that the larger the length of a complete manual is, the less usable the system (or, the harder a user finds it to reach a given level of performance). We will argue below that the length of manuals (that is, manuals of defined structure and coverage)

can be determined from the system specification, in which case we obtain measures of usability with a high degree of credibility.

2.1 User knows nothing: how long will be taken?

If we assume the user knows nothing, they can be modelled as acting as a Markov process (i.e., pressing buttons with a uniform or particular conditional probability distribution). The assumptions are particularly appropriate for walk up and use systems. Such analysis is actually quite tricky, but it does provide considerable insights. See [11].

2.2 User learns by trial-and-error: how quickly can they learn?

As well as very obvious measures (such as the probability that a randomly pressed button does something, which is proportional to the mean out degree), many numerical measures, such as self-entropy, readily suggest themselves.

Machine learning provides algorithms for robots to learn finite state machines. The performance of these algorithms (e.g., polynomial in diversity) give measures for how long a (systematic!) user would have to take.

2.3 User knows everything: but how much is that?

The all-pairs shortest path is the minimum number of button presses to get from whatever the system is doing to anywhere else. Presumably the skilled user needs to know (some) shortest paths to use a system effectively. The mean of the all-pairs shortest paths is a measure of how much a user needs to know to solve *any* problem; when multiplied by the number of states, it gives a measure of how much the user needs to know to solve *all* problems. For the few systems we have been able to compare, the value decreases convincingly with informal evaluations of usability.

The logarithm (base the number of buttons) of the number of states is the theoretical minimum achievable for this measure, and it is rarely reached.

2.4 User knows everything: is it correct?

The optimal Chinese postman tour gives the minimum number of button presses to check that a user correctly understands a system. It gives a theoretical low bound on the time taken for a usability test, and is (almost always) vastly lower than the expected length of a random test procedure (adequate user testing of a system must take years, even for simple systems).

Finding an optimal postman tour is a quite separate problem than using its length as a complexity measure. Finding the tour has polynomial complexity (for a serial algorithm), and is non-trivial in general. However, certain graphs have simple tours: a randomly Eulerian graph, for example, has a tour that can be found by pressing any button that has not yet been pressed in the current state. Clearly, the usability can be improved (if this is the task!) by the system indicating which buttons have previously been pressed. (We discussed the general issue of help above.)

2.5 User knows everything, but system states may be indistinguishable: is what the user knows correct?

This is a much more interesting problem than the foregoing, which assumed each state (of those being checked by the user) was distinguishable (indeed, distinguishable in constant time). It is a serious problem (bisimulation) that has exponential complexity, both to work out how best to do it, and to actually do it (and one can never be certain, since states that seem the same may ultimately be differentiated). The recommended approach, then, is not to (thoroughly) formulate the usability complexity given these assumptions, but rather, having identified the problem, to change the design to reduce its complexity. The Chinese postman tour, which assumes states are distinguishable, has length at most twice the number of edges in the graph, which is clearly a lot easier on the user.

The general problem of indistinguishable states (and the consequent difficulty, or intractability for the user, of prediction) comes under the domain of the WYSIWYG principle, and is discussed in [10].

2.6 System is in an indeterminate state: what does the user need to know?

One less the maximum over all states of the minimum length of cycles passing through each state is the best worst-case number of actions the user has to embark on to recover from a single ‘overshoot’ accident. (A correct undo makes this value 1.) Hence, the larger, the worse.

One may wish to design a system so that advice is, so far as reasonable, independent of the initial conditions. There is a trade-off in how to achieve this. One system we examined could only achieve advice similarity if all advice started, “Switch off”; but another we examined instead had almost as many buttons as features, and this meant that regardless of its state a button generally achieved the same-named goal. An important theoretical measure relating to this is diversity.

2.7 User reads the manual: how long do they take?

If a user is to know the most efficient way of using a system, the all pairs shortest paths reference manual is appropriate, though lengthy. (The length of this is the square of the number of states.) For non-trivial systems or non-emergency responses, however, more manual structure will be required to ensure the manual is of a manageable length.

Conventional paper manuals are trees in more ways than one: they are made up from sections, containing subsections, containing sub-subsections and paragraphs of text. Thus, a conventional manual is an embedding of the system graph into an ordered tree, typically a spanning tree of that graph. For any one graph there will be very many trees: there will be very many feasible conventional manuals. The appropriate complexity measure is the length (or weight) of a minimal spanning tree. The depth, too, is a useful measure of complexity.

The following heuristic generates a ‘minimal manual’ as a minimal spanning tree: (i) the documentation of a vertex is its label and labelled set of out edges; (ii) all edges, from vertices spanned by a subtree, with a common end vertex are documented at the root of that subtree; (iii) the number of edges is minimised. The manual is constructed from a pre-order walk of the tree; subtrees generating sections, subsections and so forth. This manual is minimised by factoring incident edge repetition. (There is an interesting similarity with such an approach and a heuristic for drawing statecharts.)

It is a routine task to identify bridges, vital edges and hinge vertices in a graph. These concepts correspond to critical concepts in manuals (e.g., without knowing about a bridge a user cannot reach a subgraph). The critical components can be weighted to be described first, or otherwise highlighted. Minimising the depth of a bridge in a manual corresponds to enabling the user to know how to switch between ‘modes’ easily; maximising corresponds more to training wheel systems [2], where it is assumed that the user should be protected from components on the far side of bridges.

The complexity measures (length or weight) of optimal manuals can be considered complexities of the corresponding systems. (Clearly, one can obtain the measures without generating the manuals!) This discussion shows that appropriate manual design depends on the user tasks one is targeting it for.

2.8 User is confused: how long does it take to recover?

At its simplest, the complexity measure here is one half that of the length of the manual complexity. But this assumes (rather, makes quite clear) that the complexity depends on (i) the system making it clear what state it is in (ii) the user having an algorithm (e.g., sequential search) for finding the correct section of the manual able to use the feedback from the system (iii) the user knowing that they are confused.

We can say, roughly: for an optimal system (independent of manual) or for an optimal manual (independent of system), the time is logarithmic. Of course, one can do better: the system might have a little display “see manual page 95” — better still, the entire manual might be embedded in the system itself. Either way, the synchronisation problem can be avoided (which introduces, at best, a quadratic slow down).

2.9 Other criteria

Complexity is not the only measure of a system’s usability. There are many consistency criteria (e.g., no out degree should exceed the button count; the graph should be strongly connected).

Suppose the system has a bug. How long would the user take to find it? We might estimate the bug density and model the user as a Markov process, from which we can find the expected number of bugs

found per user action. From there it is an easy matter, allowing for working hours, to find the actual time a user takes to hit a bug. Of course, this is a different measure from how long it takes a user to recognise a bug! Nevertheless, the time is likely to be surprisingly large, given that the density of bugs is typically low when user testing is attempted.

Other, more interesting, criteria can be established, such as the minimum number of on/off feedback indicators that ensure some visible change on any state transition. This is the logarithm (base 2) of the chromatic number. One can easily count or identify state transitions that are inadequately coloured (which can confuse the user because indicators do not change).

2.10 Visual-perceptual analysis

Although the foregoing ideas are mathematical and permit numerical comparisons to be made, this does not preclude visual analysis. It is trivial to generate dot-and-arrow drawings. Ranked embeddings are a variation where the position of a vertex is such that it shows, for example, the least number of button presses to get to it from somewhere else. Some systems are clearly lop-sided when represented like this, and this may indicate a design problem, or a trade-off that could be — or should have been — analysed.

3 A simple design question discussed

Would a VCR be improved by providing menus instead of buttons? Unlike many devices, VCRs are certainly connected to adequate output screens (TVs) that could display the relevant text.

This is a typical design question. It may be answered quantitatively.

The meaning of a VCR's buttons are approximately constant. The play button (almost always, but not invariably) makes a VCR play a tape. Thus, buttons can be used to achieve user goals without first determining what state the VCR is in, and without reading any text (or decoding any icons).

Menus use direct manipulation selection. Although a remote control could use a pointing device (probably a thumb operated joystick), the user is required to *read* a menu, and the order of items in the menu becomes problematic. The user has a search problem, a task which might be reduced in complexity by using a tree structured menu; on the other hand, as soon as a tree is used, the user needs more knowledge, since a 'selection' may be applied at either internal nodes or leaves.. But then the same job can be done with a number of buttons equal to the order of the tree, except having the advantage that a practised user need not read the buttons to operate the system.

Taking numbers from one VCR analysed: if button presses were used to select from a menu, on average one needs 14 up or down operations, then a select operation. In fact, this system has an average button press count of just under 4 presses to get from any state to any other state. Its conventional button interface, then, is not only faster (if you have learnt where the buttons are, or can find them appreciably faster than reading), but more task-oriented than a linear menu. Perceptual issues, such as the design of the menu's on-screen font (which may be awkward to read on a TV) should be offset against similar issues in button design (obscure icons, badly lit, specular reflections).

The number 4 (average button press; mean shortest all-pairs state to state path length) compares unfavourably with the information theoretic minimum of 2 which can be achieved with the 8 buttons available on this machine: there are subtle trade-offs between minimal button presses against semantically relevant button presses. This begs the question of what semantic relevance is! If we define it to be that pressing a button brings the system closer to any of a class of states that implement a function of the same name, then we can readily check that the system conforms to the criterion.

The numbers mentioned above are all easily established. They enable trade-offs to be considered rationally and quantitatively. It may be that a sensible design would provide both a menu *and* button style interface, however this design choice would have implications for the manual length. In turn, that suggests the menus and buttons should be isomorphic, which is a concept (for enough states) that will take less space to explain than enumerating the individual differences!

4 What if a system is 'too' complex?

Most interesting systems are not most conveniently modelled as graphs. A simple alarm clock viewed as a finite state machine may have over 3000 states. Even if this is reasonable for an interactive system,

it may be unmanageable.

There are several solutions:

- If the system is precisely defined, a program can be written that constructs the graph, (more-or-less) regardless of its complexity. If the system is not precisely defined, it's not obvious what the designer is up to
- We can notice that much complexity arises from pedantry in what we wish to describe. If users can be assumed to understand the numbers involved in times, then we do not need to describe every possible change of time: instead all times can be treated equivalently. The alarm clock becomes about 3 augmented states.
- Graphs can be decomposed into simpler components. (In the case of a simplified teletext TV set, a statechart factoring reduces the state count from over 300 to just over 20, which is more manageable. A difficulty is that a statechart so constructed will have orthogonality that may not correspond in detail to the real system — but this is an evaluation, not a design problem, for that TV.)

For complex systems, then, it is clear that the designer will have to exercise careful judgement in choosing appropriate levels of description. This is nothing new!

5 Conclusions

We have argued that very many direct measures of usability can be obtained at design time. These measures have to be used judiciously, but they are easy to obtain and give direct indicators of usability. Given a system definition, the measures discussed here can either be obtained by using standard mathematical library functions, or by easily written programs. It is surprising that they have not been more widely used.

Acknowledgements

Many people have suggested ideas that have enhanced this paper. The following suggested ideas when I was attentive: Stuart Anderson, Andy Cockburn, George Coulouris, Alan Dix, Steve Draper, Joachim Funke, Michael Harrison, Peter Ladkin, Andrew Monk, Prue Thimbleby, Nina Warburton.

References

- [1] BUCHNER, A. & FUNKE, J. (1993) "Finite-state Automata: Dynamic Task Environments in Problem-Solving Research," *Quarterly Journal of Experimental Psychology*, **46A**(1), pp.83–118.
- [2] CARROLL, J. M. (1990) *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, MIT Press.
- [3] ARROLL, J. M., SMITH-KERKER, P. L., FORD, J. R. & MAZUR-RIMETZ, S. A. (1987–1988) "The Minimal Manual," *Human-Computer Interaction*, **3**, pp.123–153.
- [4] DIX, A. J. (1991) *Formal Methods for Interactive Systems*, Academic Press.
- [5] DRAPER, S. W. & OATLEY, K. (1992) "Action centred manuals or minimalist instruction? Alternative Theories for Carroll's Minimal Manuals," in *Computers and Writing, State of the Art*, Holt, P. O'B. & Williams, N., editors, pp.222–243, Intellect Press.
- [6] HARRISON, M. D. & BARNARD, P. (1993) "On Defining Requirements for Interaction," *IEEE Proceedings RE'93*, International Symposium on Requirements Engineering, pp.50–54.
- [7] HARRISON, M. D. & THIMBLEBY, H. W., editors (1990) *Formal Methods in Human-Computer Interaction*, Cambridge University Press.
- [8] SHACKEL, B. (1986) "Ergonomics in design for usability," HCI'86, pp.44–64.

- [9] SKIENA, S. (1990) *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Addison- Wesley.
- [10] THIMBLEBY, H. W. (1990) *User Interface Design*, Addison-Wesley, 1990.
- [11] THIMBLEBY, H. W. (1991) "Formal Methods with no Psychology," *IEE Colloquium on Theory in Human-Computer Interaction*. IEE Publication No. 1991/92, pp.6/1-6/6.
- [12] THIMBLEBY, H. W. (1992) "Heuristics for Cognitive Tools," in NATO ASI Series F, *Proceedings NATO Advanced Research Workshop on Mindtools and Cognitive Modelling, Cognitive Tools for Learning*, Kommers, P. A. M., Jonassen, D. H. & Mayes, J. T., editors, pp.161-168, Springer Verlag.
- [13] THIMBLEBY, H. W. (1993) "Combining systems and manuals," *Proceedings BCS HCI Conference*, in press.
- [14] THIMBLEBY, H. W. & WITTEN, I. H. (1993) "User Modelling as Machine Identification: New Methods for HCI," *Advances in Human- Computer Interaction*, **IV**, Hartson, H. R. & Hix, D., editors, pp.58-86.