Design Aloud: A Designer-Centred Design (DCD) Method

Harold Thimbleby
Middlesex University
LONDON, N11 2NQ

harold@mdx.ac.uk

September 30, 1998

Abstract

Designer-centred design methods build on the knowledge and skills of the designers to improve user interface design. Unlike users, designers are trained to handle complexity and to make design trade-offs. They are the people centrally in control of the design process. Designer-centred design complements and refocuses user-centred design methods. A particular designer-centred design approach, "Design Aloud" is introduced and justified.

Computer systems do not often perform as well as they could do. The reason for this is sometimes blamed on the lack of a user-centred approach to design. Designers concentrate on features and system performance and, indeed, often the user is ignored. In reaction to this, user-centred design is promoted (Landauer, 1995): orient the design process to understand the users, collect empirical data on users' behaviour and performance, and iterate designs to improve them. Quite rightly, usability has been emphasised in HCI; Monk *et al.* (1993) even say that close involvement with users is the *only* way to produce a good design. By studying how users really perform, usability of systems can be significantly improved, and engaging users in the process, as a political act, builds their commitment to the design. Once one adopts this view, it is seen to have many problems (e.g., expert users may find it hard to articulate what they do) and it has become an active area of research and progress. There are many techniques such as 'think aloud' (Lewis, 1982) that are available to inform the usability process; see Baecker, Grudin, Buxton and Greenberg (1995) for a review.

Whilst such methods are necessary, they emphasise the two opposite ends of design — requirements and final product — rather than the designers themselves, who are the people best placed to influence the process of design for the better. It is true that many 'designers' may in practice be programmers with little experience of real design: but this is not how it should be. The method described here will help programmers become better designers.

There are many design-centred approaches, such as: QOC (questions, options and criteria): MacLean, Young, Bellotti and Moran, 1991; Design Rationale: Lee and Lai, 1991; — this list is not exhaustive. A recent method, Contextual Design (Beyer and Holtzblatt, 1998), is a holistic user-centred approach. Contextual Design is 'customer-centred,' aiming to find out what customers want, negotiate with them in design teams, and hence deliver more finely targeted systems. It provides well-defined steps suitable for ISO9000 compliance. Throughout, the emphasis is on being customer-led, facilitating customers to contribute to the

design. This is certainly necessary, and goes part way to redress the usual disdain systems development has for users; but it is not sufficient. It loses out on potential technical contributions to design. The users may not know best, and merely computerising their tasks may be computerising inefficient and inappropriate procedures. Beyer and Holtzblatt suggest that the designers' intuitions cannot be trusted to produce useful systems. This seems far too narrow, even cynical! It is more realistic to say that design is complex and that humans — whether users, customers or designers — are not capable of doing perfectly. Some information from users is invaluable, and designers should heed it; likewise — and hardly ever emphasised — designers' intuitions are valuable and surprisingly powerful. Designers have more experience, a system-wide view of design, training in handling complexity, and so forth. One should also draw on their expertise: hence *designer*-centred design.

Except for completely trivial designs, no usability study can explore more than a few of the possible behaviours (the number of which may be infinite). All usability methods (task-based analysis, contextual inquiry, think aloud) are useful, but at some point you need *designer*-centred design. I believe that design*er*-centred design is a useful focus, sharper than design-centred design. Designs are passive things (and often fixed once they exist); designers active, committed, trained, reasoning agents.

Design processes that involve users, what they think and what they do (whether with mock ups, prototypes, manuals, or field systems, *etc.*), can only be informed by what is encountered, rather than by gestalt of the design itself. Fortunately designers have implicit knowledge about design, as it were, the *intentions* of the design, whether they have in fact been realised in the current version of the implementation. By talking aloud about these design intentions, they become explicit, and (even if already explicit) they have to be justified. The effort to the designer of making design features explicit and clear to a user indicates their elegance.

We therefor suggest a technique very similar to think aloud, or participatory design, but intended to elicit direct principled guidance for design improvements. Instead of a *user* thinking aloud while performing a task, a *designer* thinks aloud performing an explanation of the design to a user. Hence, *Design Aloud*.

It is best if the designer is the system programmer — we are fully aware of the cultural hurdles to be overcome in involving programmers in HCI! (Our notion of *designer* here is someone with significant technical understanding and thorough understanding of the implementation.) Thus, it is a technique that helps the designer discover design problems; it uses the designers' perspective rather than the users' perspective. (Or, to use the current jargon, it exploits the designer's ontology.) Unlike other design-centred approaches, Design Aloud does not try to find out what the design is, or how it came to be, but to get the designer involved in explaining and justifying the actual system in use. It is not a walkthrough to find out what the user thinks (although this is valuable as another design tool), but a technique to facilitate the designer to think. Some of the designer's explanations will be convoluted, thereby identifying potential areas of change to the design that would then be able to be more clearly expressed. Some parts of the design may be elegant and satisfying; these features may be good opportunities to generalise so that the same elegance or style is available more uniformly or consistently across the system.

The designer explains to a potential user how the system works and can achieve various tasks. The designer must explain the system's principles. This should not be a demonstration, and certainly not a demonstration that follows a script: demonstrations are too easily turned into drama that encourage people to imagine more there than there is (Thimbleby, 1996; Schrage, 1996) — as in the theatre, it is easy to forget the artificial scenery and that the actors are only acting — and in turn, an exercise of imagination makes it very hard for the design to be completely honest. Webb (1996) makes a similar point: we need a design approach that recognises the difference between a real system and a show. Wishes too easily become fantasies. A central purpose of designer centred design is to uncover wishes and hence iterate designs so that unfulfilled wishes become realities. Preparation of a session will merely help conceal defects.

It follows that Design Aloud is better suited to certain sorts of user interface, and moreover to certain sorts of implementation approach. It does not help so much with, say, "multimedia" systems — where the underlying interaction has already been fixed, but where design is concentrating on content and emotional affect. For systems like consumer products (video recorders, etc.) there is a superficial conflict between having a working system and being able to make any use of the results of Design Aloud — the solution is to use appropriate design tools, perhaps so that a device emulator is available and so suggested changes and improvements to a working design can be effected. Without an emulator, Design Aloud would be restricted to systems that were already committed to hardware: that, in this case, Design Aloud would be ineffective is a comment on the poor design practice, rather than the technique itself. Finally, designers may not know best: getting a system to work at all is often quite sufficient an achievement, that doing it

well is secondary to doing it at all — in this case various techniques for delaying commitment should be employed (Thimbleby, 1988; 1990a).

In Design Aloud it is more productive not to use a mock-up system (e.g., ones made out of paper, storyboards and other simulations). A running prototype can be used, but it should be treated as the real system, or the designer has to be very careful to maintain integrity. The advantage of working with computers is that they have higher standards of accuracy than humans; using a prototype in Design Aloud does not exploit the computer's high standards on the system that matters — a prototype is not the delivered system, so doing Design Aloud on a prototype misses many important insights that relate specifically to the very system the users will use.

We want the designer to be able to say "I used the wrong algorithm," "the same algorithm works over here too" or "the algorithm is more general than that" — these sorts of computationally-insightful comments about the design cannot emerge from a storyboard that has no algorithms! Design Aloud generates insights into system design, and details of system design, rather than to those more superficial aspects such as appearance and functionality. Typically a Design Aloud session generates interesting programming ideas. Thus, Design Aloud is used when a system is nearly ready for delivery, but before its design is fixed. Judging when this is will be very tricky for products running under commercial pressure. However, Design Aloud is ideal for when a new feature (or group of features) is added to an existing system: this makes the Design Aloud session more focused. Since most modern systems are extremely complex, making it unlikely that one designer is in control of the whole system, Design Aloud can be much more effective with the extra focus when dealing with small, component, changes to systems. (It is *still* likely that changes to *earlier* parts of systems may still be uncovered!)

Typically a recording of the session is made. The designer is aware of the recording, and may either have a button to mark the recording tape (for ease of future searching) or can make announcements like, "I must fix that" so that the session can proceed rather than get stuck on just initial problems. Designers may also like to keep a notebook during the session. Use of the notebook can enable the designer to note the design insight and then more quickly move on to explore other features in the design.

The rules require that the designer explains how the system *really* works, honestly, and explains any limitations. The designer can also express intentions: they can describe features as they wish them to be — but they must be strictly honest in distinguishing actual and wished-for features.

Simple example: The designer explains that the user's name must be typed in a box on the screen. They (being honest) say that the name must be no more than 100 characters. They may also say to the user, "can't do that" — i.e., some part of the system is not finished yet! That is an obvious usability bug, but in Design Aloud no user has had to discover it. For the sake of brevity, these are trivial examples. Ideally, the designer in a Design Aloud session would think of new theorems and algorithms underlying the system design; subsequently implementing those in the system would make it better — if not empirically better with respect to measurable usability, it would certainly make the system better in terms of reliability, maintainability, and in very many technical ways that would support easier iterative design.

Following Carroll (1990) and Thimbleby (1990a), Design Aloud assumes that the design is improved by changing it so that the designer's explanations are simplified or shortened. Hedges (e.g., "only if less than 100 characters") can be eliminated — by making the system tolerant of longer strings, or by making the system gracefully handle its limitations. Design Aloud will be less useful, or have to be used differently, for games development.

Background to the technique

The road to wisdom? Well, it's plain
and simple to express:
Err
and err
and err again
but less
and less
and less.

Piet Hein (1966)

As Francis Bacon put it: if you want to err less, you need to know what the errors are.

Since hearing it in 1982, I have been impressed with Knuth's insight that good program design is literature. He built (and I copied) a system for literate programming (Knuth, 1992). If design is literature, it is but a short step from this to find tools that make design worth talking about.

I have often used Design Aloud, but not thought of it as a technique as such until I was explaining a language I had implemented to Saul Greenberg. The language had conditionals but no logical operators; I explained that this wasn't necessary since any expression could be represented by appropriate conditionals. A moment's reflection led me to realise that introducing operators would simplify the explanation of the user interface (although at the cost of increasing the user interface feature count).

I have used a "silent" variant of the technique extensively in my own work: instead of spoken Design Aloud, the designer writes down careful explanations — that is, detailed and thorough user manuals.

I have not attempted an evaluation of its effectiveness compared with other techniques. It is possible, then, that the claimed effectiveness has more to do with me and my idiosyncrasies than with design in general. That Design Aloud makes sense to me is not, in itself, sufficient reason to promote it more widely! I hope that this paper will encourage others to evaluate the technique and circumscribe better than I can its contribution to effective design. But I don't think design can be so easily evaluated. As a designer, I find the technique helpful for the sorts of activity I engage it. That is sufficient. And the investment in trying the idea is low, so one does not need to be persuaded by (inevitably limited) empirical results on other systems.

Thimbleby (1990a) and (1990b) give many examples of where Design Aloud might have worked, had the method been used. Thimbleby (1990b) is a discussion of bugs some of which are admitted by system implementors in systems' manuals. Here, the user manual is a sort-of transcript of a Design Aloud session explaining to a user how the system works. Most user manuals quietly admit many bugs, which perhaps would have been better dealt with by programming them away — and not needing to explain them. Literate programming and literate using (Thimbleby, 1986; 1997) are a very convenient way of interleaving manuals and programs so that designers have an easy choice: make a system simpler, or explain its complexity.

Thimbleby (1987) gives an extended example designing a terminal package, where the 'recorder' for the designer-centred sessions was the paper itself. That paper also provides discussion of selecting design principles and prioritising design decisions between them when they are in conflict. Thimbleby (1997) gives a brief discussion relating to the design of a web authoring tool.

Designer-centred design helps redress some of the overly-user centred approaches to design, which actually dis- or re-orientate technical designers, programmers, away from being concerned with use to just implementing what is elicited from users. Because implementation is hard, and because user-centred approaches (iteration, *etc*.) generate many and rapid demands (and "the customer is right"), designers rarely have a chance to think constructively about the design. Many systems have huge numbers of users, have very plausible collections of often highly visible features, but are nasty systems that have little design coherence. Designer-centred design will help empower programmers to take more initiative. This will be of particular benefit for systems that have unknown or unpredictable classes of users —— systems for use on the world wide web being a good example.

Programming is very hard; we have this on the authority of one of its most prominent and skilled practitioners (Knuth, 1996). Most programmers are probably happy enough to have got a system to work, and have no spare energy to make it, additionally, work well for users. Moreover many proponents of user-centred design also trivialise programming (e.g., Landauer, *op. cit*.) into a passive following of empirical insights: users drive design, and programmers try hard to make users' ideas reality. These two forces together create a culture where the designers' potential insight into usability never gets nurtured. I hope Designer-centred design helps reinstate designers to a pivotal and active role in design.

Conclusions

Design Aloud does not centre on users or their tasks, but on the intentions and plans of the designer; it is principle-led. We do not obtain statistics on how often users encounter problems (e.g., with conditionals), but what we do is to identify principled improvements to a system design that apply systemically. A metaphor will make the virtue of this apparent. Suppose we use conventional empirical usability to test the quality of a wooden floor. Most users will most of the time not get their feet caught in the one raised splinter. Therefore the floor is sold with a flaw. On the other hand, by using designer-centred design, any competent designer would say "watch out for the splinter!" Having got the designer to think aloud explaining the floor to users, the identification of areas for improvement is easy. Clearly, as the metaphor makes clear, designer-centred design is most appropriate in safety and mission critical systems design — and in systems where quality and elegance of design is important. It is notoriously difficult to get programmers and technical designers, for whom the technique is aimed, committed and involved with

HCI; here is a technique that can lead to very satisfying design insights, and which may help promote HCI in areas where change can be effected.

Acknowledgements

Ann Blandford and Saul Greenberg made many valuable suggestions that improved the idea and this paper enormously.

References

R. M. Baecker, J. Grudin, W. A. S. Buxton & S. Greenberg, editors (1995) *Readings in Human-Computer Interaction: Toward the Year 2000*, 2nd. edition, San Francisco: Morgan Kaufmann Publishers, Inc.

H. Beyer & K. Holtzblatt (1998) *Contextual Design*, San Francisco: Morgan Kaufmann Publishers, Inc.

J. M. Carroll (1990) *The Nurnberg Funnel: Designing Minimalist Instruction for Practical Computer Skill*, Cambridge, Massachusetts: MIT Press.

P. Hein (1966) *Grooks*, Cambridge, Massachusetts: MIT Press.

T. Landauer (1995) *The Trouble with Computers*, Massachusetts: MIT Press.

J. Lee & K-Y Lai (1991) "What's in a Design Rationale," *Human-Computer Interaction*, **6**(3/4), pp251–280.

C. H. Lewis (1982) *Using the "think aloud" method in cognitive interface design*, IBM Research Report, RC–9265, Yorktown Heights, New York: IBM.

A. MacLean, R. M. Young, V. M. E. Bellotti & T. P. Moran (1991) "Questions, Options, and Criteria: Elements of design space analysis," *Human-Computer Interaction*, **6**(3/4), pp201–250.

A. F. Monk, P. C. Wright, J. Haber & L. Davenport (1993) *Improving Your Human-Computer Interface: A Practical Technique*, London: Prentice-Hall.

D. E. Knuth (1992) *Literate Programming*, CSLI Lecture Notes, **27**, Stanford: Center for the Study of Language and Information.

D. E. Knuth (1996) *Selected Papers on Computer Science*, CSLI Lecture Notes, **59**, Stanford: Center for the Study of Language and Information.

M. Schrage (1996) "Cultures of Prototyping," in *Bringing Design to Software*, edited by T. Winograd, J. Bennett, L. De Young & B. Hartfield, p200, New York: Addison-Wesley.

H. Thimbleby (1986) "Experiences with Literate Programming Using CWEB (A Variant of Knuth's WEB)," *Computer Journal*, **29**(3), pp201–211.

H. Thimbleby (1987) "The Design of a Terminal Independent Package," *Software—Practice and Experience*, **17**(15), pp351–367.

H. Thimbleby (1988) "Delaying Commitment," *IEEE Software*, **5**(3), pp78–86.

H. Thimbleby (1990a) *User Interface Design*, London: Addison-Wesley.

H. Thimbleby (1990b) "You're Right About the Cure: Don't Do That," *Interacting with Computers*, **2**(1), pp8–25.

H. Thimbleby (1996) "Internet, Discourse and Interaction Potential," in L. K. Yong, L. Herman, Y. K. Leung and J. Moyes, editors, *First Asia Pacific Conference on Human Computer Interaction*, pp3–18.

H. Thimbleby (1997) "Gentler: A Tool for Systematic Web Authoring," *International Journal of Human Computer Studies*, **47**(1), pp139–168.

B. R. Webb (1996) "The Role Of Users In Interactive Systems Design: When Computers Are Theatre, Do We Want The Audience To Write The Script," *Behaviour and Information Technology*, **15**(2), pp76–83.