Understanding user centred design (UCD) for people with special needs

Harold Thimbleby

Future Interaction Technology Laboratory (FIT Lab), University of Swansea, Wales
harold@thimbleby.net

Abstract. "User centred design" (UCD) has become a central, largely unquestioned, tenet of good practice for the design of interactive systems. With the increasing recognition of the importance of special needs in influencing design, UCD needs to be re-examined, in particular to be clear about the difference between using its *methods*, which may not suit special needs, and achieving its *objectives*. This paper introduces a simple two-category classification of special needs, to which UCD applies very differently and which are heavily affected by developments in technology; in other words, the role of UCD, particularly with respect to special needs, will continue to change and demand close scrutiny.

1 Introduction

User Centred Design (UCD) is one of the essential concepts in Human Computer Interaction (HCI), interaction design, usability engineering, interaction programming — and all the other fields that have designing effective interactive systems as their goal. User centred design is design that is based around the real and actual requirements of users, and typically involves task analysis, prototype development with users, evaluation, and iterative design.

Designers unaware of UCD typically argue they do not need user centred methods: they can use their systems, users don't complain ... what's the point? Users may not complain for all sorts of reasons, for example that the user interface defects are too complex for them to understand, and their managers or other pressures to achieve mean that they are more interested in working around problems and succeeding than in giving designers feedback. UCD clearly has a cost, and if the benefits are minor, then there is no business case to do it. It is no wonder, then, that there have been many forceful arguments for UCD arguing against these attitudes. Classic arguments in the literature include Landauer's *The Trouble with Computers* [21], Gould and Lewis's 1983 paper "Designing for usability—key principles and what designers think" [9, 10], and Draper and Norman's *User Centred System Design* [25]. However, the arguments for user centred design emerged from the 1970s historical context and, in particular, as a reaction against what might be called technology centred design (TCD?) as it then was — and as it was then failing. These high-profile, persuasive arguments, and many others, established UCD as epitomising the field. ISO standards, such as ISO 13407: *Human-centred design processes for interactive systems* have established UCD as a definitive standard.

There is surprisingly little hard evidence that UCD works reliably or as well as is generally supposed; for example, in a panel session at the ACM CHI conference as recently as 2006 [27] the question debated was whether think aloud (a widely used UCD technique) works. The panel pointed out that most arguments for the efficacy of UCD methods are based on usability work where the aim is to improve a system, rather than to test any method, so the evaluation of the method will be confounded: the experiments are really about usability, not usability methods. That is, the purpose of usability work is to improve a system, then empirical work may just deliver what people want, namely an improved system, rather than a tested UCD method. When

Change since 1970s	1970s	2000s
Rise of consumerism	Employees required to use bespoke systems	Consumers buy what devices they like
Appropriation	Computers restricted to work environments, and had to be used for work in standardised ways	Gadgets appropriated by users for tasks they were not designed for (e.g., phone-as-a-torch; SMS)
Standardised office systems	Systems had to be designed for specific users new to computers	Employees have transferable skills
Standardised business processes	Every business works differently, typically using ad hoc paper-based systems	Since office and other systems (e.g., email, web, mobile phones) are well known, they define business practice
Complexity of design	UCD could evaluate typical systems	Some systems too complex too rely on user evaluation
World wide web	No international or universally available resources	Huge resources with standardised user interfaces (browsers)
Users with special needs	Few employees had special needs; even fewer used computers; users were homogeneous	User needs are very diverse; UCD per user is very costly
Ubiquitous computing	All computers (or terminals) are desktop	Many physical forms, wireless, etc available
Customer loyalty	Users do not buy or own interactive systems	Making systems different makes users less able to use competing products
Accessibility software	No allowances for special needs	Standardised accessibility software (e.g., text to speech; magnification) widely available
Change in usability	Usability is effectiveness, efficiency and satisfaction (ISO 9241)	Usability includes empowerment, enjoyment, experience, enchantment, care, socialisation
Public interfaces	No public interfaces	Walk-up and use interfaces everywhere

Table 1. Representative changes affecting UCD since the 1970s. There are many more factors we do not have space to cover: changing demographics (many more elderly users); social interaction (web 2.0, massively multi-player games); physicality; complex consumer devices (e.g., home cinema, sewing machines); location-aware (GPS etc); games consoles; mobile phones; point-of-sale (POS) devices; iPod; legal requirements ... and so on. Not all changes are positive.

looked at more closely there is very little experimental study specifically supporting the use of UCD methods, though there is an *enormous* literature on improving systems!

When there is experimental evidence in support of UCD, great care must be taken to interpret the results: for example, UCD methods to reduce design defects are typically very different from UCD methods aimed at improving the user experience, and in any case many "UCD" experiments might better be understood as psychology experiments aiming to understand human behaviour rather than design processes as such. In particular, generalising from methods that have been shown to work with normal users is unlikely to be reliable with special needs users yet, ironically, special needs users would generally benefit even more than normal users by successful UCD work.

Almost all UCD studies (or development using UCD methods) has been performed on "normal" users, and indeed often undergraduate students who, for instance, generally won't have cognitive problems. Taking account of special needs in UCD does matter: a study of think aloud with blind users showed it had unexpected problems [6]. Ironically, while remote evaluation promises access to larger pools of users — and hence the opportunity to recruit adequate num-

bers of test subjects with specific special needs — studies show that special precautions are required [26] (a paper which is to be recommended for its principles). In short words: UCD is certainly not a fixed set of techniques that work reliably, regardless of the user population.

These issues are all *before* we start questioning the quality of the science behind UCD. The Gray and Salzman papers [11, 12] discredited — in some people's eyes — the quality of many usability experiments; Cairns [4] questions the use of statistics; and Chen [7] questions fundamental psychology assumptions — he argues it is an artifact. On the other hand, science is a progression towards consensus, and it does not have to be perfect all the way; the point of papers is not to be perfect unassailable science, but to argue that certain ideas are worth holding on to. And one can argue in many ways; science is just one way, statistics another, and starting from human values another. UCD is about respecting users, and it needs no science to argue the value of that.

We need first to see why UCD makes sense, rather than trying to test it and showing it does when it does (if it does). For the more that UCD makes sense, the less we need to rely on experiments that are anyway very hard to do in such a complex area. For the more UCD makes sense, the more able we as designers and agents of change can adapt it to the needs of particular cases. This is a pertinent point of view for design for special needs.

Instead of relying on arguments for UCD, instead it is more appropriate to say UCD is a *value*: we believe that respecting users and their diversity is important, and moreover that we do not begin to understand the needs of users until we explore with them what these needs are. "Focus early on users." If we do not take account of accessibility and inclusion, we are by default excluding some users. Instead, we believe UCD is a self-evident good and that is sufficient to take it seriously. It would be nice if there was some rigorous experiment supporting that view, but the complexity of real life — to say nothing of our willingness to believe even the most tenuous experiments supporting UCD because we *want* to believe them! — means that this is an unlikely dream. In the context of this paper, rigorous experiments supporting UCD in the special needs domain are going to be even more problematic (consider experimental controls, statistical power are all compromised, to say nothing of more complex ethical clearance issues). Instead, we need to understand what UCD is about and what it tries to achieve, rather than ask for hard evidence.

This paper will argue that UCD, while remaining a self-evident good, is no longer a universal panacea to user interface design. Rather, we need to understand what UCD aims to achieve, and then to work out how to achieve those (or better) goals for the problems in hand. There can be nothing more ironic, surely, that saying "let's be user centred," and then deploying a standard UCD technique regardless of the specific users and tasks and expecting UCD to deliver design value.

There are several reasons why UCD should no longer be given the same status as it justly deserved in the 1970s: the rise of consumerism, the increased complexity of user interfaces, the increased familiarity of users with computers, and the increasingly important role of users with special needs. . . , and many other reasons, as illustrated in table 1. (In table 1 and throughout this paper, we use "1970s" as an informal but convenient label for a style and approach to design that might be said to have peaked during the 1970s period.)

What is very clear from table 1 is that the role of interactive systems has changed enormously since UCD was first proposed; moreover, some of the assumptions that UCD entails, such as the effectiveness of evaluation are called into question because the world has changed. UCD will, of course, change further in the future: the emphasis is moving from desktop to mobile, from mobile to ubiquitous, from ubiquitous to embedded. Similarly, the underlying concept of "usability" is moving from productivity to satisfaction.

In the 1970s, evaluation of a pool of users would provide statistically useful data cost-effectively — for example, an hour's evaluation with 10 users would provide high quality in-

formation to guide iterative development of a design. Classic papers such as Nielsen and Landauer's mathematical model of finding usability problems [24], and many UCD methodologies such as cognitive walkthrough [34], think aloud [6, 27], and scenarios [28] emerged around this time. However users with special needs have diverse requirements, as for instance there are fewer similarities between users, and statistical power of evaluation studies with one user is nonsense. More importantly, in the 1970s environment, the effort required for a UCD study could be shared amongst many users, whereas design with special needs in mind may put all the evaluation effort onto few users or even a single user. In some cases, the special needs of those users may mean that sufficient data is too costly to obtain: they may be too exhausted to continue, they may have limited attention, their carers have additional needs, and they may resist cooperating or wish to make the experience counterproductively more exciting (say, if they have a low mental age). Paradoxically, despite such obvious problems with UCD, special needs users (together with their friends and carers) have more to gain from well-designed user interfaces.

Not only is UCD harder with designing for single users, but modern interactive devices are much more complex than typical 1970s systems that were the staple of UCD. A "5 user" heuristic analysis will not have time to explore enough of a system to identify design issues. Thimbleby [31] reports a simple device where typical UCD would take months to have 50% chance of finding a crucial error. Special needs users may not have that time or attention available, and as there is more variance between special needs users, the Nielsen and Landauer formulæ are too optimistic.

2 Objectives of UCD

The prime objective of UCD is to improve the usability of delivered interactive systems, and while the definition of "usability" is changing, and can change for different applications (e.g., compare games, where usability is about fun, flow and engagement, with medical devices, where usability is about reducing untoward clinical incidents), the objective breaks down into uncontentious subsidiary objectives:

Define and prioritise usability values with users — otherwise known as early focus on users. Surely the purpose of doing anything is to help people (perhaps sometimes just ourselves)? The first step is to identify who these people, the users, are and how they might best be helped — what their values are. UCD takes it for granted that we do not know *a priori* how best to help users without their participation.

Match task requirements to design What is wanted and what can be achieved (given the timescales for delivery, limited resources, politics, and so forth) are rarely the same! In general, the design should realise the realistic requirements of the users' tasks, baring in mind that the task that can actually be supported by the system may not be the same as what the users would ideally like.

Remove defects from the design, and from the requirements It is very unlikely that a design will work well to start with. UCD emphasises that designs have defects and that they should be found. Less often emphasised is that requirements have defects, partly because users did not understand how a working system would change their behaviour. UCD realises that there are different sorts of defects and different costs in identifying them. So-called "killer defects" should be found efficiently and quickly.

Test against usability criteria Defects are things wrong with the design; usability is about how the design supports the usability criteria. For example, can tasks be completed within appropriate times?

Iterate design to continuously improve Typically the previous stages of UCD identify issues that can be fixed, however fixing them creates a *new* system with *new* issues.

> Realistically, iteration is tightly interleaved with evaluation in a process of continual improvement — this aims to avoid evaluating parts of a design that may be modified during iteration.

In short, make usability a goal of design and find ways to achieve it, given the reality of an imperfect world where neither requirements nor delivered systems are ideal. UCD really requires delivered systems to be continuously improved through further user evaluation and feedback to designers; many business models rely on this, continually producing new versions of their systems.

Seen like this, UCD as a concept applies equally to special needs and to "average" users. As will become clear, UCD has to be achieved by using methods, which have costs, and we must not confuse the methods with the goals. A caricature would be: "did you use UCD?" – "yes, I used think aloud and a user experience questionnaire with Lickert scales and a χ^2 test ..." – "were any of your users dyslexic?" – "does it matter?" In particular, we will see that UCD has costs *on the user*, and when the user is special needs these costs cannot be spread around a large community of homogeneous users, and may thus be disproportionate to the assumed gains of the conventional UCD effort.

3 Why UCD was invented out of the 1970s

Again, taking "1970s" as a rough category, prior to the 1970s there were three sorts of user. There were people with hands-on use of the computers; these were usually called "operators." There were people who programmed computers; called "programmers." And, thirdly, there were the people who thought of themselves as "users," but who worked through the operators rather than using the computers directly themselves. Operators were highly trained. Often, users were the operators and programmers — with the consequence that designers and users had the same conceptual models. Finally, the nature of the technology meant that most users had very little real interaction with the computer: a typical task might take a day to work through. Computers tended to be used for "batch" tasks like payroll, where what might be called the "interaction cycle" took a month, and most users did nothing explicit to get results from the system.

As the 1970s progressed, the power of computers increased and more users got direct hands-on experience with increasingly fast turnaround times.

The most prominent style of interactive system was the mainframe computer (or minicomputer) connected to many terminals or user workstations. Software was written for the mainframe, and ran identically on each terminal. Typically, a business would tender for their processes to be automated, and a software company would write new software to do so. Most of the programmers who wrote the software would be new to the task that was to be automated, and typically they would write software to do what managers wanted — managers, after all, were their clients, not the end users — rather than to support what users were actually doing. Typical early-1970s tasks were word processing, airline reservation, air traffic control, stock control: cases where there were few designers working for many users — and users mostly with very different social backgrounds to the designers.

If the software company was lucky, they would find more businesses with similar processes, and they would sell their software into these businesses. Now the connection between the software design and the users was even more tenuous!

Most designers have considerable development experience. Inevitably, much of their experience was based on designing systems that they used. Most computer science courses today continue this: most education assumes students are individuals and are assessed alone. When people design systems for themselves (or systems for assessment that only they will use, even if the assessment asks for systems designed for other people), the designer is the user, and therefore

— notwithstanding bugs in the system — the user model is the system model. The designers learn, if not correctly, that there is no need for UCD.

The key idea — the beginnings of UCD — was captured by Hansen's 1971 slogan "know the user" [13], which UCD takes to mean two things: identify the user, and identify what they want. This can be refined further: what people really want may not be what they say they want, and what people want will change as soon as they get something, and there are conflicting types of user — for example, the employer or carer may have wants that are different form the actual users. Knowing the user isn't just knowing about the user, it is knowing who the user is; the client who pays for the design work is rarely the user who benefits most directly from a successful system.

"Knowing the user" is often unpacked as knowing the user's model. The user model as an idea was not popularised until Newman and Sproull's revised edition of their *Principles of Interactive Computer Graphics* [23] the then leading, and almost only, graphics textbook had a chapter on user interface design that discussed the user model at length. The user model is supposed to be how the user conceptualises their work practices. Since designers are not users, and have often had a training that emphasised technological expertise, in many cases the designer's conceptualisation of the user model would be inaccurate. Furthermore, programming is hard and even if a designer had a good conceptualisation of the user model, they would be unlikely to realise a computer program that matched the user model. The first problem is addressed by studying the user's tasks, and thus constructing an explicit, reality-based user model. The second problem is addressed by iterative design: build a system then see how well it works, then use the evaluation to improve it. This is basically Gould and Lewis's advice: early focus on users and tasks; empirical measurement of product usage; iterative product design.

The "user model," while a pithy phrase with an intuitive meaning and an apparently straightforward and central role in UCD, begs many questions. What *exactly* is a user model? Arguably, that question led to two different research programs: the model human processor (Card, Moran & Newell [5]), and explicit user models such as ACT/R (Anderson [1]). Neither have found wide use in interaction design, and neither have been used for special needs work.

A very important role for UCD is to decide what the goals of a design are, and to make these goals explicit, otherwise the design will optimise unstated (and possibly unknown) goals. In the 1970s, the general assumption was that usability was to be equated with productivity or effectiveness — but effectiveness for whom, though? In the 1970s effectiveness was usually defined in business or employment terms. Now, usability is defined much more in a user-centric way: what is the user experience (UX), what do they want? In the case of special needs UCD, it is crucial to decide whether the purpose of the design is rehabilitation, augmentation, fun, education, socialisation, persuasion (e.g., to take medicines), and so on; these are very different goals. In the 1970s, it would be clear that the purpose of UCD for a special needs user was to make their work more productive; today, it is recognised that there are many other valid goals of design, and, moreover, that the goals are not mutually compatible. For example, enabling a worker to be efficient in the work environment may be at odds with fun or socialisation, and there is clearly a difference between a user interface that is efficient and a user who is efficient — one requires conventional UCD, the other requires persuasive techniques [8], and this "conflict" becomes apparent in issues like security (e.g., users need to be persuaded to use passwords effectively). However, a user may decide they wish to increase their earning potential or self-esteem by working more productively — so even "work goals" may be aligned to user goals.

In the 1970s, few system implementers were trained in design or HCI, and many because of their ignorance of UCD may have thought they knew enough to design well [19]. "Hey it works well for me!" Worse, without formal evaluation (which UCD assumes), they would be influenced by the survivor effect: when implementers or designers hear from current users, these must be

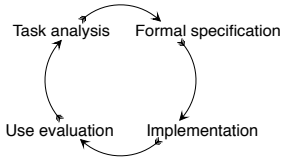

Fig. 1. Iterative design requires going around the design cycle, here shown schematically, *more than once*, otherwise there is no iteration! Therefore *every* step, not just the obviously UCD steps, are necessary for usability.

users who have survived using their system. The ones that failed no longer use the system, so the designers get a biassed sample, which appears to confirm their success at designing good systems.

4 Why UCD is not *sufficient* now

Central to UCD is of course working with the user. The user performs tasks and the performance (speed, enjoyment, error rate and so forth) assessed; errors and impasses in the design are identified — for example, a think aloud protocol may have the user saying "I don't know what to do now," and this point identifies an opportunity for the design to be clearer. Unfortunately, there is a tradeoff:

- If the system is complex, the rate at which UCD identifies worthwhile improvements to design is too low, particularly if there are too few users available to work with the system. A complex system means that UCD insights may not be reliable; they may not generalise across the design — parts of the design may have escaped scrutiny. This means the cost to the designer of UCD is too high.
- If the system is simple, it is likely the design is simple because the user has physical or cognitive capacity problems. This means the cost to the user of UCD is higher.

Obviously this tradeoff is a gross simplification of the complexity of real users and real design, but it suggests that there is a "sweet spot" where a system is not too complex for UCD to be reliable and the user capable enough to be able to provide sufficient data. The position of the sweet spot varies with the user. In other words, UCD is not always best — or, certainly, the cost/benefit tradeoffs that are implicit in most UCD (e.g., in Nielsen's satisficing cut-off of 5 users [24] is probably too high) need to be carefully reconsidered when the up-front costs to the users is higher.

A more detailed look at interaction design (H. Thimbleby & W. Thimbleby [33]) shows that design is not just concerned with the user; the program has its own problems too! A successful design needs both sides to work (and, of course, to work together). This requires competent software engineering, and it may involve solutions such as better user training where the device cannot be configured to match the user's requirements. Thimbleby & Thimbleby introduced the terms "externalist" (the external-to-device side, the user and environment, the UCD) and "internalist" (the internal-to-device side, the hardware, the program, the SE) to help emphasise that neither is sufficient but both are necessary; see figure 1. One could do internal work that contributes to a good design but with no UCD — and, conversely, one could do good UCD work that contributes to a good design but involving no internal work.

5 Why UCD is not *necessary* now

Many (such as Gould and Lewis [9, 10]) would argue that UCD is essential. UCD can often improve designs, but sometimes the cost is out of proportion to the benefits, particularly when the cost to the user cannot be spread between more than one user. It may be far easier for the user to choose a different product than to do UCD on a developing product. Interestingly, Gould & Lewis do not justify UCD (or, more specifically, their four principles): they take them for granted: their classic paper is not evidence for the principles, but evidence about how little they were known by practicing designers.

Despite the rational arguments for (and not entirely for) UCD outlined above which expose its assumptions, many would defend UCD uncritically. They may be right (and this paper wrong); they may be right but subject to unspoken assumptions; or, possibly, they may be certain but misguided. The latter is a significant possibility. UCD is hard, both for the users and for the designers. People who have gone through a UCD experience are likely to find design insights, but after hard work. Cognitive dissonance predicts [29] that they are then likely to justify the UCD against the suggestion that better results could have been obtained more easily. Similarly, the users involved in the UCD exercise may prefer the device because they themselves put a lot of work into its UCD, whether the device is "objectively" improved or not. Of course, if a user thinks they prefer a design that is not objectively better, then thinking they prefer it means they *actually* prefer it.

We could distinguish between ideal UCD and realistic business UCD. In the ideal UCD case, eventually the user gets a very nice product that does what they want; the problem for business is that this takes a long time to recover development costs. Instead, a knowingly-imperfect system is sold, and then UCD leads not to iterative design but to *new* products the user pays for. This creates market churn, and most everybody is happy: users get nearly usable systems faster, business has an income stream, but the environment suffers as systems become obsolete rather than updated through actual iterative design. One of the issues is that a business (or business sector) will hold a portfolio of products and, over time, they will drop ones that are too costly to improve; these become not just obsolete but unavailable.

In this business model, the goals of UCD are sort-of satisfied, and users pay for it, and the longer they stay in the market paying for improvements the better the devices become. Some people have old mobile phones, because for them the cost/benefit of newer, improved models is insufficient to upgrade. Unfortunately, special needs users are never a large part of a market particularly for devices that have low cost because they are massively mass produced (like mobile phones). Thus the business section may drop a model or style that some users would prefer to stay with. For example, as mobile phones have become smaller and lighter, which the mass market prefers (which statistically speaking UCD prefers), users who require large keys and large displays (for legibility) lose out.

6 The reality of bad design

Heathrow's Terminal 5, which failed immediately after it opened, *was* user tested, but users were involved at such a late stage that the timetable for opening could not be modified [35]. Or consider the Abbott AIM infusion pump, a device involved in a fatal drug overdose, with such a poor user interface that a very brief UCD study identified many flaws [15]: but the device is in production, and changing it would involve expensive recalls, retraining, and the possibility of incompatible devices (original and improved) left in use and operators making transfer errors.

The root cause analysis of the fatality [15] suggested that the hospital should undertake human factors studies (i.e., UCD without the designers!) itself, to choose which devices to procure. This

view stems from the authors of the report apparently believing that when a device behaves as designed it is correct, though as they acknowledge that the study showed problems in use, they blame not the design (or the design process) but the choice of device, the hospital training, and so on. If the hospital had procured the device after its own user studies, it would have had one that was easier to use. There is no moral case to be made that the manufacturers need not have done adequate UCD themselves (in fact, the design defects, so easily uncovered in a brief study, are horrific), but it is an argument that a successful business case can be made for not doing UCD if official reports suggest users should do them for you.

In a better world, an informed manufacturer would identify usability criteria and the performance of their products against those criteria — this would be a direct output of in-house UCD. Thus users (or procurers) could then choose in an informed way between various products against their own usability and other performance criteria.

7 The cost of iterative design

The purpose of UCD is to improve design, but the very act of doing a UCD study involves the user working with the previous design. The more data is obtained about the design that is supposed to be being improved, the more the user (or users) are learning about the obsolete design. For a mass produced device (like a typical mobile phone) the number of users involved in the UCD study is a tiny proportion of the user population of the improved device, so UCD is worthwhile: most people get a better deal. But in user interfaces for special needs users, there are very few — often, just one — user and the benefits are much less obvious. If the user is cognitively impaired, the effort of working with the prototype may mean that improving the design would cause transfer errors, transferring the prior training on the prototype to the modified design. Ironically it is likely that the most deliberate and conscious activities during evaluation are around the worst parts of the design exploration, and hence the most effort was put in by the user into the very parts that are likely to be changed. If the UCD studies take a long time, perhaps the best improvement is no improvement at all!

One solution to the cost of revision problem is to perform UCD on sketch prototypes [3], so that no or limited learning can really take place, but the designer can still get useful evaluation.

8 The UCD slogan may be counter-productive today

Interactive system design is not just about designing (and deploying) effective interactive systems, but it is also about advancing the field: innovating, spreading best practice, and developing the expertise and skill set of practitioners and researchers in the field. Publication is one of the standard ways of disseminating advances; to the extent that the community insists that a "proper" publication or contribution to knowledge must include UCD, then some valid contributions will not make it through the refereeing process. In an ideal world, one might do everything perfectly, but — and particularly when working with special needs users — doing UCD as well as innovating may be one step too much for a typical contribution. Furthermore, if UCD is to be done as well, then a publication may take more than a year to prepare, and then it misses the natural annual cycle of publication in conferences.

UCD (evaluation, statistics, controlled experiments, and so forth) are desirable in any paper claiming to make advances to design knowledge, but they are only one form of arguing validity.

Here is an example of this problem, and how it detrimentally affects the wider user (and research) community. Beveridge and Crerar [2] reported on a user interface designed for stroke victims, which had been evaluated on just three aphasics. At the time, they had been unsuccessful

in getting continuation funding to develop their work because referees wanted a user study with more subjects. It is of course very hard to find other stroke victims with comparable issues for which the same user interface innovations would be relevant. That was 1999; today, with the accessibility of the web, it is possible that more users could have been recruited: the world as a whole probably has thousands of users who could benefit.

9 Two sorts of special needs: programmable & unprogrammable

There are two sorts of special needs in users, that can be addressed by very different solutions. Computers are programmable and can meet any "virtualisable" requirement (and in the future, with nanotechnology and other developments, not just virtual but any physical requirements too). However programmability comes at the cost of increased complexity (whether or not it is easy to use). Thus the two sorts of special needs are those that can be handled adequately by programmability and those that cannot. This is not a classification of special needs as such, but a classification of special needs and systems *together*, as the following contrasting examples make clear:

- An example of a special need that is handled well by programmability is a patient in a hospital, for instance in an intensive care unit. The patient has very specialised needs, but the complexity of the system they are using (e.g., an infusion pump) is managed by trained clinicians: exactly the same system can be used for many different patients.
- An example of special needs not handled well by the flexibility of programming is a user with dementia, or a user in a wheelchair trying to use an "automatic" toilet — without a carer, both users have to handle the complexity of the programmable user interface.

Some special needs are moving from one sort to the other, as technology develops. Thus although mobile phones are getting smaller and therefore harder to use for some, they are also getting more programmable: programs could change a phone to have large buttons and large writing — and, crucially, the incremental cost of programs (their manufacture and sale) is negligible compared to the hardware: special needs users in this category will benefit enormously. (Apple's iPhone is an example.)

"Programmability" is a spectrum; it can range from completely free device programmability, where the programmer can achieve anything the device is capable of; user programmable, where the programmability is restricted (and perhaps therefore safer); configurable, or set by the user choosing from a limited set of choices; or adaptive, where the device learns and modifies its behaviour itself (e.g., [32]). The spectrum therefore covers reprogramming that might involve taking the device or system to a specialist (e.g., to change the firmware), to something that happens automatically without explicit intervention. Note, of course, that for some users, adaptibility may be unwanted: its consistency over time may be important.

10 Best practice...

UCD is not an automatic way to improve the user experience, particularly when the user has special needs (or the user is a carer of a person with special needs). The earlier parts of this paper explored the rationale for UCD, and has shown how the design environment has moved on from the classic 1970s assumptions and will continue to develop. UCD is still relevant today, but best practice must not use it unthinkingly. It is a tool for better design, and it is not the only tool in the box. The conference series of *Computers Helping People with Special Needs* is an excellent resource [16], illustrating the value and diversity of the field.

A short paper cannot capture "best practice" in just a few words, but given the discussion has been about UCD and the reader of this paper can be confidently assumed to have a significant background in design practice and its (vast) literature (not all of it sympathetic, e.g., [36]), the following points may be flagged as important components of contemporary best practice:

Understand how to think about UCD The usual way of thinking about UCD at a higher-level is to view it from the maturity lifecycle perspective [18]: people (or organisations) don't have problems with usability (because they don't know about it) ... through to UCD is properly embedded in the organisation (or in a designer's repertoire) as a key part of development and evaluation. While the maturity model is helpful, it tends to take "UCD" as a fixture, whereas this paper has argued that UCD itself has to be understood, not just used as if all users and tasks were essentially the same and all amenable to the same bag of techniques: we can always improve processes, and while maturity models focus on the use of given processes, such as using UCD, the activities (here, UCD) themselves can be improved.

Get involved with real users as soon as possible The key lesson of UCD is iterative design is essential to make systems better — but the later UCD is engaged, the more investment there has been in possibly suboptimal design. The earlier UCD can be used, the more flexibly and usefully designers can respond to the UCD insights. Thus, use field research, sketches, non-functional prototypes, focus groups and other methods that do not rely on things "working." Holzinger gives a good example of the issues and value of working with paper prototypes with elderly users for the design of a mobile questionairre [14].

Use indirect methods (early lifecycle methods) It seems tautologous the UCD involves users, but it need not use them directly, and therefore the costs of UCD need not be borne by the final users themselves. For example, scenarios can be developed and these portray very effectively to designers even though the scenarios may have been created with other users (for example, there are many available scenarios of elderly users, and these can also be used to help inform new designs — thus reducing the impact of UCD on users). Book, films, digital stories, forum theatre and many other techniques play a similar role, in addition to the standard repertoire from UCD: expert walkthrough, heuristic evaluation, guideline checklists.

Use analytic methods Much of the work conventionally done with users can be done analytically, whether on the human side (e.g., using a simple Fitts Law analysis [22] to cognitively sophisticated simulations such as CogTool [17]), on the system side [30], or both [20].

Use best practice software engineering A significant cost in UCD is defect elimination. Many defects are a result of poor programming practice, and can be avoided in principle.

Test and evaluate with simulations Simulated users can be used to obtain much use data that can be used in UCD; moreover, *defining* the simulated user (robot or virtual user) requires studying the users and tasks carefully, and this may obtain useful *explicit* design insights. A single simulated user can be used repeatedly in many UCD trials, thus saving effort for the real human users.

Simulated users can range from purely random [30], based on large survey data, to sophisticated models, such as those based on realistic cognitive models like ACT/R [30].

Make a range of well-defined products/prototypes If users can choose between products, then they can choose those that best suit their needs. This assumes that manufacturers are honest about their products' capabilities, and (particularly for special needs users) it assumes that manufacturers allow their products to be trialled. Many devices can be simulated on the world wide web; these should be as faithful to the real device as

possible (which doesn't readily happen [31]); many devices could be semi-realistically simulated on PDAs or other handheld devices, for more realistic assessment.

Make systems programmable, adaptable or end user programmable You either have to make a system good enough first time or adaptable. It does not matter whether a device is programmable by specialists (as opposed to the manufacturer), by the user, or is self-programmable (otherwise called "adaptive") or controlled by "preferences" the user (or carer) defines — but making a device programmable means that the user or user community or even enthusiasts can adapt the device to specific needs of users.

As mentioned above, this list is not exhaustive. For authoritative ideas, see the relevant ISO standards, such as 9126, 9241, 13407, 18529 and TR 16982.

11 Conclusions

UCD is important but is often confused for using the conventional methods that achieve "UCD" for normal users. When users have special needs, they are more diverse and the costs of performing UCD increases dramatically, particularly for the intended end-users. Ironically, for some special needs, the very act of UCD may train the user (or carer) or focus their expectations in a way that will limit the (conventional) scope of UCD and iterative design. The standard tradeoffs (e.g., five users for a heuristic evaluation) are all called into question when the cost/benefits are so different, and when the cost/benefits cannot be amortised across a large population of similar users.

The nature and role of UCD has changed considerably since it was first introduced in the 1970s period, and it continues to change. In particular there is an increasingly clear distinction between user needs that can be met through programmable and non-programmable methods on particular systems: this is a sufficiently clear distinction for it to be used in classifying special needs and tasks for the purposes of interactive system design. An example is a programmable mobile phone where the typography can be controlled by graphics programming so visual problems might be worked around; on the other hand, graphics programming cannot compensate for physical keys being too small or not providing adequate tactile feedback — on different hardware, or for different tasks, what special needs can be "programmed around" will be different.

The purpose of this paper has been to encourage the designer, particularly the special needs designer, to achieve the *aims* of UCD without unthinkingly simply using its current *methods*, thinking the means justify the ends. The purpose of UCD is to respect users, and to find ways to make that a practical possibility during the design process and hence to obtain benefits for everyone.

References

1. J. R. Anderson and C. Lebiere. *The Atomic Components of Thought.* Lawrence Erlbaum Associates, 1998.
2. M. Beveridge and A. Crerar. A multimedia presentation system for the remediation of sentence processing deficits. In *Proceedings of INTERACT'99*, pages 272–280, 1999.
3. B. Buxton. *Sketching the user experience.* Morgan Kaufmann, 2007.
4. P. Cairns. HCI... not as it should be: Inferential statistics in HCI research. In L. J. Ball, M. A. Sasse, and C. Sas, editors, *Proceedings of BCS HCI 2007*, volume 1, pages 195–201, 2007.
5. S. K. Card, T. P. Moran, and A. Newell. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, 1983.

6. S. Chandrashekar, T. Stockman, D. Fels, and R. Benedyk. Using think aloud protocol with blind users: A case for inclusive usability evaluation methods. In *Assets'06: Proceedings of the 8th international ACM SIGACCESS conference on Computers and Accessibility*, pages 251–252, New York, NY, USA, 2006. ACM.
7. M. K. Chen. Rationalization and cognitive dissonance: Do choices affect or reflect preferences? Technical report, Yale School of Management, 2008.
8. B. J. Fogg. *Persuasive technology.* Morgan Kaufmann, 2003.
9. J. D. Gould and C. Lewis. Designing for usability—key principles and what designers think. In *CHI'83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 50–53, New York, NY, USA, 1983. ACM.
10. J. D. Gould and C. Lewis. Designing for usability: Key principles and what designers think. *Communications of the ACM*, 28(3):300–311, 1985.
11. W. D. Gray and M. C. Salzman. Damaged merchandise? A review of experiments that compare usability evaluation methods. *Human-Computer Interaction*, 13(3):203–261, 1998.
12. W. D. Gray and M. C. Salzman. Repairing damaged merchandise: A rejoinder. *Human-Computer Interaction*, 13(3):325–335, 1998.
13. W. J. Hansen. User engineering principles for interactive systems. In *AFIPS Conference Proceedings*, volume 39, pages 523–532, 1971.
14. A. Holzinger, P. Sammer, and R. Hofmann-Wellenhof. Mobile computer in medicine: Designing mobile questionairres for elderly and partially sighted people. In K. Miesenberger, J. Klaus, W. L. Zagler, and A. I. Karshmer, editors, *Computers Helping People with Special Needs (ICCHP'06), Proceedings of the 10th International Conference, Lecture Notes in Computer Science*, volume 4061, pages 732–739, 2006.
15. Institute for Safe Medication Practice Canada. *Fluorouracil Incident Root Cause Analysis*, 2007. www.cancerboard.ab.ca/NR/rdonlyres/D92D86F9-9880-4D8A-819C-281231CA2A38/0/Incident_Report_UE.pdf.
16. International conference on computers helping people with special needs conference series. In *Lecture Notes in Computer Science*, volume 2398, 3118, 4061, etc. Springer Verlag, 2002 on.
17. B. E. John and D. D. Salvucci. Multi-purpose prototypes for assessing user interfaces in pervasive computing systems. *IEEE Pervasive Computing*, 4(4):27–34, 2005.
18. T. Jokela and T. Lalli. Usability and CMMI: Does a higher maturity level in product development mean better usability? In *CHI'03 extended abstracts on Human factors in computing systems*, pages 1010–1011, New York, NY, USA, 2003. ACM.
19. J. Kruger and D. Dunning. Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments. *Journal of Personality and Social Psychology*, 77:1121–34, 1999.
20. X. Lacaze, P. Palanque, D. Navarre, and R. Bastide. Performance evaluation as a tool for quantitative assessment of complexity of interactive systems. In *Proceedings of the 9th International Workshop on Interactive Systems, Design, Specification, and Verification (DSVIS)*, volume 2545, pages 208–222. Springer, 2002.
21. T. Landauer. *The Trouble with Computers.* MIT Press, 1995.
22. I. S. MacKenzie. Movement time prediction in human-computer interfaces. In R. M. Baecker, W. A. S. Buxton, J. Grudin, and S. Greenberg, editors, *Readings in human-computer interaction*, pages 483–493. Kaufmann, 1995.
23. W. Newman and R. Sproull. *Principles of Interactive Computer Graphics.* Mcgraw-Hill, 2nd edition, 1979.
24. J. Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings ACM CHI'92*, pages 373–380, 1992.
25. D. A. Norman and S. W. Draper. *User Centered System Design; New Perspectives on Human-Computer Interaction.* Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 1986.
26. H. Petrie, F. Hamilton, N. King, and P. Pavan. Remote usability evaluations with disabled people. In *Proceedings of ACM CHI 2006*, pages 1133–1141, 2006.
27. J. Ramey, T. Boren, E. Cuddihy, J. Dumas, Z. Guan, M. J. van den Haak, and M. D. T. D. Jong. Does think aloud work? How do we know? In *CHI'06 extended abstracts on Human Factors in Computing Systems*, pages 45–48, New York, NY, USA, 2006. ACM.

28. M. B. Rosson and J. M. Carroll. *Usability engineering: Scenario-based development of human-computer interaction.* Morgan Kaufmann, 2002.
29. C. Tavris and E. Aronson. *Mistakes were made.* Harcourt, 2000.
30. H. Thimbleby. *Press On: Principles of interaction programming.* MIT Press, Boston, Massachusetts, 2007.
31. H. Thimbleby. User-centered methods are insufficient for safety critical systems. In A. Holzinger, editor, *USAB'07—Usability & HCI for Medicine and Health Care*, volume 4799, pages 1–20. Springer Lecture Notes in Computer Science, 2007.
32. H. Thimbleby and M. A. Addison. Intelligent adaptive assistance and its automatic generation. *Interacting with Computers*, 8(1):51–68, 1996.
33. H. Thimbleby and W. Thimbleby. Internalist and externalist HCI. In *BCS HCI 2007 Conference*, volume 2, pages 111–114, 2007.
34. C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practictioner's guide. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, 1994.
35. E. Widdup. We told T5 bosses the system was a farce... they ignored us. *London Evening Standard*, page 17, 9 April 2008.
36. D. R. Wixon. Evaluating usability methods: Why the current literature fails the practitioner. *Interactions*, 10(4):28–34, 2003.