The design of software to support creative practice

Colin Beardon
Exeter School of Arts and Design, University of Plymouth

Abstract
There is general acceptance of the need to include 'the user' in the process of design. The design of software for creative practice involves software designers treating product designers as 'users' and it is argued that neither party has a clear understanding of this relationship. It is also argued that software for creative practice is better seen as a language, rather than as a task-oriented product. As such it should establish a code and allow users to make interesting statements within that code.

The 'Visual Assistant' is new applications software that has been developed over the past three years to support performance education. Fundamentally, it involves the manipulation of 2D images within a 3D space. A set of design objectives for the product are described. The design method used in development is also described. The process of evaluation is not clearly defined in relation to such applications and the general procedure whereby users' experience is fed into evolving design is discussed in the light of the experience at three teaching workshops.

Keywords: creativity, language, software design, theatre, users

Designers and users

Few designers, be they software designers or product designers, would disagree with the maxim that the design process should involve 'the user'. Better products will emerge if the needs of users are discovered and if the user is involved in the testing and evaluation of an emerging product. So far, so good: but the method whereby the experience of users is included in the design and development of a product has been the subject of much debate and is in need of updating in the context of contemporary multimedia computing (Bjerknes and Bratteteig, 1995).

As a software designer now working in the field of Art & Design I am acutely aware that the word 'design' covers many disparate activities. Software design and product design (I will use this latter term loosely to cover many areas of creative practice as taught in our colleges) may seem formally similar, but they diverge widely in their concerns and methods. For the past three years I have been involved in the design and implementation of software to support creative practices and, as such, have been involved in the interesting juxtaposition of (software) designers developing technology for (product) designers to use. The product designers are thereby put into the role of 'users', while the software designers need to enter into dialogue with these other designers. What emerges from such involvement is not only greater understanding of the two design traditions, but also new software design methods relevant to the domain of creative practice.

Software design methods can often seem very formal and in areas such as safety-critical systems they need to be. Many computer-based systems are not of this kind, however, and there is now a long tradition of more user-centred approaches to software and systems design. Enid Mumford was an early champion of user participation through the socio-technical approach (Mumford and Henshall 1979). Mike Cooley and others argued for a human-centred approach to design with increased recognition of the role of tacit knowledge (Cooley, 1980; Rosenbrock, 1989). Pelle Ehn and others have pursued a more theoretical and, at times, more political agenda (Ehn, 1980; 1999). Many of these approaches blur the boundaries between the software product and the wider context of use.

Usability testing is a more product-oriented procedure within software design that addresses the viewpoint of the user (Dumas & Redish, 1994). Essentially, it involves a group of typical users testing a design at each stage through the performance of real tasks.

The problem, from our perspective, of usability testing so described is that it applies primarily to a field of work very different from the experience of creative practitioners. There are two underlying assumptions behind Dumas' and Redish' work which may be true for a large number of software applications, but which do not hold in Art & Design. The first is that computers are only used to perform clearly defined tasks; typically to transfer information unambiguously from sender to recipient with the minimum of information loss. There is no recognition that the computer might be used in conjunction with imagination or in play. The second is that computers are used primarily to convey information encoded as letters and numbers within a limited two-dimensional plane. There is no recognition that multimedia computing generates subtleties of meaning which can be harnessed to great effect.

If software designers are not good at paying attention to the needs of creative practitioners, it also the case that artists and designers do not make very good 'users'. Typically they engage with pre-packaged software, proud of the fact that they use it as "only a tool" — by which they seem to mean that they accept no responsibility to understand or improve upon it (a strange connotation of the concept of "tool" within creative practice!) This phenomenon — the refusal to accept the role of critical tool-user — is further complicated by the way in which many creative practitioners actually benefit from software. During preliminary investigations undertaken in 1995 (Beardon et al, 1997) we discovered that many artists and designers are particularly contrary users of software. They will often say that they only got interesting results from software once something went wrong. Some elevate this to a technique, forcing the software to misbehave or deliberately subverting the intentions of the software designer in order to get an interesting response. Though this is anecdotal, I am surprised just how many times I have heard the chance remark that progress was only made when software did something unexpected.

With software developers locked into an isolated world, with designers-as-users refusing to engage in constructive criticism of the tools and with creative users deliberately trying to subvert the intentions of software designers, is there any hope for a more considered and genuinely useful generation of software for those involved in creative practices?

Software design objectives

A key principle in re-thinking software design for creative practice was the idea of language as the key to understanding work and its context. In a famous maxim Ludwig Wittgenstein rejected his earlier, formalist understanding of language and said, "Don't look for the meaning, look for the use." His exploration of 'language-games' was an attempt to understand language in the context of practical work in the real world. As a design methodology, this can be reversed: by examining uses of language at work we can better understand working practices and, thereby, design better products. The problem in the case of art and design is that language is not primarily text or speech based. Following some experiments with developing iconic languages, the decision was taken to embark upon the development of a major piece of software with the intention of producing not a task-oriented product but a 'language' within which interesting things could be said.

Earlier work on the 'Virtual Curator' posed the question, "What does software mean?" (Beardon & Worden, 1995). There are formal answers to this question but the meaning we are concerned with is not determined by the software designer or the computer or the end user but is negotiated between them. This is particularly the case when any element of creative practice is involved (and, as far as I am concerned, all significant jobs involve some creative practice). Creativity is the *search* for

meaning or for the expression of meaning and the meaning therefore cannot pre-exist the practice.

For the past three years I have been working on software called the 'Visual Assistant'. It is intended primarily for people working in the theatre and concerned with performance, though it may have many other fields of application. It is an attempt to explore the boundaries between two-dimensional and three-dimensional representation. In essence, it enables users to import or draw 2D graphical objects and then to manipulate these objects within a 3D space. The environment has a certain logic, but it is not based upon photorealistic geometry. The uncompromising lack of photo-realism has been found by educators not to be a weakness but a strength of the software. It forces student users to express ideas approximately, yet in a way that creates what one practitioner has described as 'atmosphere' (Figure 1).

The software application is written in 'C' for the Apple Macintosh and the current version contains about 20,000 lines of source code. There is a free, downloadable version of the software at the project web site <www.esad.plym.ac.uk/va/>. The Visual Assistant (VA) will run comfortably in 10 Mb of RAM and output worlds in VRML 2.0 format, enabling students' work to be accessed over Internet and seen on different platforms (Figure 2).

The software was designed for users who are primarily interested in the theatre and are probably not at all interested in computers. Many years' experience of working with such students led me to some initial design objectives for the software.

- It must be very simple to learn: you should be able to see someone else using it and then confidently use it yourself.
- It must be simple to use: there is little point in providing 96 options when most users will only use 12.
- It should give meaningful results quickly: there is a tightly iterative process of production and evaluation.
- What happens should be like sketching: it should not matter if work is irretrievably transformed or destroyed — you can always try again.

Figure 1 Using the Visual Assistant to visualise a performance

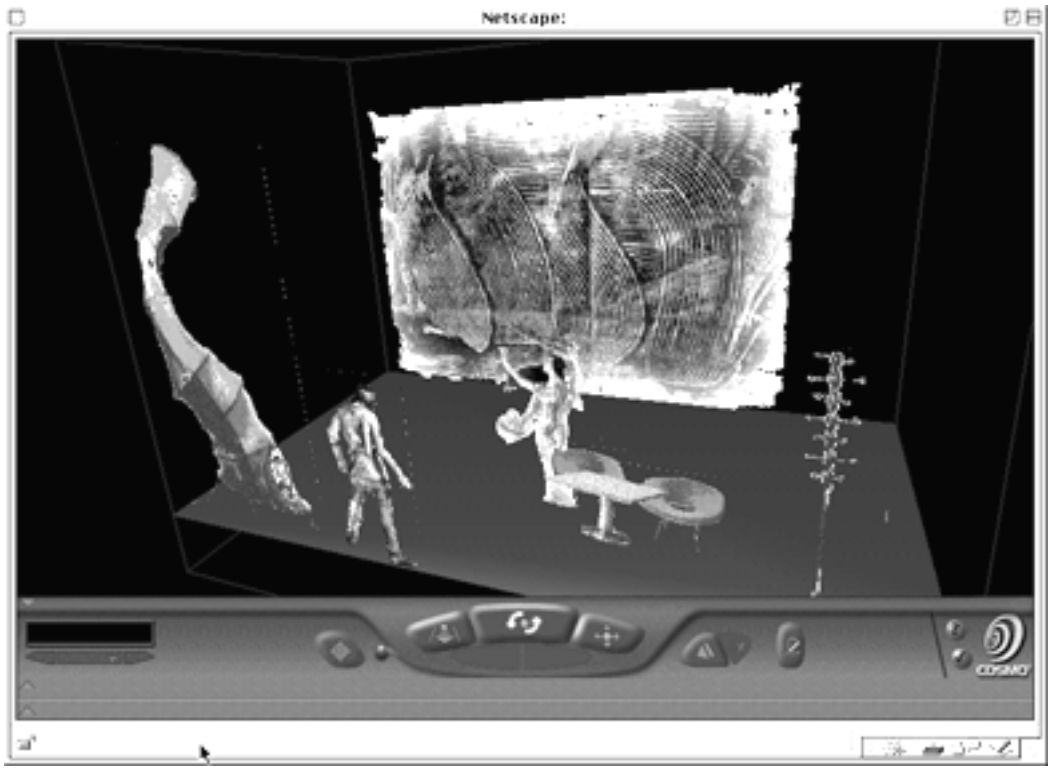

Figure 2 Viewing a Visual Assistant VRML file in a web browser

- It should support 'process' rather than 'product': particularly processes that lead to clearer understanding and better actions in the real world.
- It should present a believable 'language-game': when acting on the computer screen you should be thinking as a theatre person.
- It should support person-to-person dialogue; it should function as a common sketchpad to support critical discussions.
- It should be able to lead to more detailed implementation.

The VA also has roots in the 'alternative' or 'appropriate' technology movements, so the physicality of the product and its accessibility have always been important. In contrast to contemporary commercial software, the VA software is compact (around 500K on disk) and the ability for it to be passed around on a floppy disk is symbolic of a more personal, low-tech solution. It is a statement against the commodification of software; an important issue but one beyond the scope of this paper.

Development & Evaluation

If software is not designed around the performance of a particular task, then much of the methodology of evaluation proposed by the HCI community is difficult to apply. A new field needs to be addressed: techniques for the evaluation of software designed for creative practices. What kinds of results do we expect from users? How can we set up evaluation tests? Might we find benefits where we did not expect to find them?

I cannot pretend to have answers to these questions, but the description of the software product as a 'language for creativity' gives some clues. According to Umberto Eco, the 'open work' both establishes a code and then says something interesting within it (Eco, 1983). A 'language for creativity' should similarly enable the comprehension of the code and allow the expression of interesting ideas.

The development of the software and its evaluation must go hand in hand. The meaning of the software product is not determined in advance, it will only be determined though use. The development method therefore involves creating sample products and seeing how they are used and

given meaning. We build upon the greatest areas of success and need, and reduce those areas which are least successful. Like any living language, it evolves to match the community's needs.

I refer to this development and evaluation process as 'Put-it-on-the-table design'. By this I mean that the software designer considers design objectives and context and then comes up with a piece of working software. He or she then 'puts this on the table', with the implication that no explanation is given. It is as if the designer were to say, "There it is, pick it up and use it and I will watch. It now belongs to you and all my intentions and desires for it must be repressed. It is you, the user, who will determine its meaning". This will happen only crudely at first, and the designer must try to see long term possibilities within a few faltering steps.

Early versions of the Visual Assistant were used by members of the HaMLET project, principally educators and professionals associated with the Theatre Academy, Helsinki and the University of Paris III. At this stage I, as software designer, still had a tendency to think of the software as a device for prototyping stage designs. I was soon put right. Unknown to me, the HaMLET project coordinator used the VA to make a project progress report to the administrators in Brussels!

In February 1998 the VA (v.0.3) was used at the University of Plymouth with a group of 36 first year Theatre and Performance students. Immediately, a major problem with the 'Put-it-on-the-table design' method became apparent. As the software designer, I wished to say very little about how it should be used but my colleague, Terry Enright, acting as tutor, guessed that it would be best used to design real stage sets. Students came along with definite ideas about how their chosen play should look and immediately experienced frustration at not being able to realise them photorealistically. Nevertheless, many interesting observations were made (Beardon and Enright, 1999). For example, setting students the task of visualising famous works (e.g. *Hamlet*) was less successful — there were too many preconceptions — whereas unfamiliar works (e.g. Strindberg's *A Dream Play*) proved fertile ground for this kind of exploration. Providing users with a library of images was helpful in coping with the poor level of technical support (making the software very self-contained) but discouraged students from going out and finding fresh images.

In October 1998 I was invited to deliver a workshop using the VA (v. 1.0) at Malmö University College in Sweden to a group of 18 students. This was a very different environment: in Plymouth each student had 6 hours timetabled for this work, in Malmö it was one week and we were able to use the VA as a form of communication between six groups. Many interesting discussions took place in the theatre where we could project our VRML models, discuss their implications and manipulate them in real time. There was a very positive sense of technology in the background supporting a higher level of intellectual discourse about the realisation of the play.

In February 1999 we ran another workshop with 40 students in Plymouth,. We were now much more aware of the true nature of the product we are creating. The VA is developing a meaning in the context of performing, directing and educating. There are two specific problems that performance-based courses experience with new students. The first is that we live in a very televisual culture in which focus has been reduced to a close-up shot. In theatre there is no close-up and there is a need to see performance from outside and from a distance. The second is the dominance of words in education. Drama is often reduced to literature, advice on performing is given in words. Faced with these realities, the VA is ironically becoming a language for helping students re-learn spatiality, but to do so within the context of improvisation. As a language game it is beginning to achieve some purchase.

References

- Beardon, C. and Worden, S. (1995) 'The virtual curator: multimedia technologies

and the roles of museums'. In Barrett, E. and Redmond, M. (eds) *Contextual media: multimedia and interpretation*, MIT Press, Camb, Mass, 63-86.

- Beardon, C., Gollifer, S., Rose, C. and Worden, S. (1997) 'Computer use by artists and designers: some perspectives on two design traditions'. In Kyng, M. and Mathiassen, L. (eds) *Computers and design in context*, MIT Press, Camb, Mass, 27-50.

- Beardon, C. and Enright, T. (1999) 'The visual assistant: designing software to support creativity'. *CADE'99 Conference*, University of Teesside, 5-14.

- Bjerknes, G. and Bratteteig, T. (1995) 'User participation - a strategy for work life democracy?'. In Trigg, R., Anderson, S. I. and Dykstra-Erikson, E. (eds) *PDC'94 Proceedings of the participatory design conference*, Chapel Hill, NC, 27-28.

- Cooley, M. (1980), *Architect or bee?*, Hand and Brain, Slough.

- Dumas, J.S. and Redish, J.C. (1994), *A practical guide to usability testing*, Ablex, Norwood, NJ.

- Eco, U. (Trans by Cangoni, A.), (1983), *The open work*, Harvard University Press, Camb, MA.

- Ehn, P. (1988), *Work-oriented design of computer artifacts*, Arbetslivcentrum, Stockholm.

- Ehn, P. (1999) 'Manifesto for a digital Bauhaus'. *Digital creativity*, 9, 4, 207–216.

- Mumford, E. and Henshall, D. (1979), *A participative approach to computer system design*, Wiley, Chichester.

- Rosenbrock, H. (ed) (1989), *Designing human-centred technology*, Springer-Verlag, London.