The Right Way to Think About Software Design

Theodor Holm Nelson
Distinguished Fellow
Autodesk, Inc.

WHAT IS WRONG with people's imaginations? Writing in 1989, after designing interactive software for over twenty-five years, I am dismayed at the dreariness of the interactive software that people today think is liberating and forward-looking. Compared to what it should and will be, today's interactive software is wooden, obtuse, clumsy, and confused. The pervasive lack of imagination and good design is appalling.

I explicitly include the Macintosh in this critique, because many naive people treat that machine (especially its system software and conventions) as some important step in civilization. The premise of the Macintosh and the Macalikes is fine. We want ease of use and clear visualization of our work. But the execution and control structure are slow and drab.

To see tomorrow's computer systems, go to the video game parlors! Go to the military flight simulators! Look there to see true responsiveness, true interaction. Compare these with the dreary, pedestrian office software we see everywhere, the heavy manuals and Help Screens and Telephone Support. The world of work at computer screens is still a benighted world. And even when individual programs are tolerable, the task of making combinations of programs fit together is hopelessly arcane.

A lot of programmers and committees are creating a lot of bad designs, and the ways they are thinking about it and going about it assure that design will stay bad for a long time to come. Perhaps worse is that almost

no one seems to be able to recognize *good* design—except users, and that only sometimes.

I would like to say some things about bad design, then talk about the principles of design in general.

Elements of Bad Design

Mistake 1: Featuritis and Clutter

"Featuritis" is a principal and well-known disease of software. In the older days, featuritis meant having to remember (and correctly arrange and type) numerous keyboard options. Today, however, the blight of featuritis is taking on new forms. In the popular iconic world, it becomes a new style of screen clutter.

You face a screen littered with cryptic junk: the frying pan, the yo-yo, the bird's nest, the high-button shoe. Or whatever. You must learn the nonobvious aspects of a lot of poorly designed screen furniture and visual toys: what they actually do, rather than what they suggest. You must explore the details of each until you understand what it "really" means. In the old days, you tried to understand the various input commands and their maze of options. Today, you try to understand what the icon means. Instead of "you don't understand computers," it's now "you don't understand the metaphor."

Featuritis doesn't mean that a program aspires to do too much; there need be no limit on how many distinguishable things a program can do. The disease of featuritis is the unclarity and confusion that results from *having too many separate, unrelated things to know and understand.* The relationship between *power and flexibility for the user* and *simplicity at the interface* need not be inverse. Ultimately, the best software design assimilates all its functions to a few clear and simple principles.

Mistake 2: Metaphorics

A tedious array of icons is supported by the new Metaphoric Ideology, deeply a part of the Apple-oriented world. A metaphor is an implicit comparison, or in the more ethereal versions of the doctrine, it is a free-floating idea with a handle (such as a picture or catchword).

I would like to venture that this "metaphor" business has gone too far. Slogans and catchphrases are all very well, and these things have their uses for people who are going to learn software by *approximating* rather than *understanding.*

Let us consider the "desktop metaphor," that opening screen jumble that is widely thought at the present time to be useful. First developed at Xerox PARC, it is now seen (in different styles) on the Mac, the Amiga, the Sun, the NeXT, and the infamous Microsoft Windows. Though "look

and feel" details may vary, the underlying concept is the same, and the concept is where the weaknesses lie.

Why is this curious clutter called a desktop? It doesn't *look* like a desktop; we have to tell the beginner *how* it looks like a desktop, because it doesn't (it might as easily properly be called the Tablecloth or the Graffiti Wall).

The user is shown a gray or colored area with little pictures on it. The pictures represent files, programs, and disk directories that are almost exactly like those for the IBM PC, but that are now represented as in a rebus. These pictures may be moved around in this area, although if a file or program picture is put on top of a directory picture it may disappear, being thus moved to the directory. Partially covered pictures, when clicked once, become themselves covering, and partially cover what was over them before.

We are told to believe that this is a "metaphor" for a "desktop." But I have never personally seen a desktop where pointing at a lower piece of paper makes it jump to the top, or where placing a sheet of paper on top of a file folder causes the folder to gobble it up. I do not believe such desks exist; and I do not think I would want one if it did.

Now, I am not criticizing the idea of having mnemonics, or of creating useful visualizations in order to present ideas. In no way do I advocate the old typed command line. What I object to is severalfold: first, these mnemonic gimmicks are not very useful for presenting the ideas in the first place; second, their resemblance to any real objects in the world is so tenuous that it gets in the way more than it helps; and third, which is by far most important, *the metaphor becomes a dead weight.* Once the metaphor is instituted, *every related function has to become a part of it.*

The visualizations become locked to some sort of continuing relation to the mnemonic. It becomes like a lie or a large government project: more and more things have to be added to it, and they have to be in some way, possibly some obscure way, consistent. (A hideous failure of consistency is the garbage can on the Macintosh, which means either "destroy this" or "eject it for safekeeping.")

Here is the problem with metaphors: you want to be able to design things that are *not* like physical objects, and the details of whose behavior may float free, not being tied to any details of some introductory model. Metaphors are like WYSIWYG: useful in limited contexts, but ultimately a drag, a dead anchor [See Nelson, 1989].

The alternative to metaphorics is *the construction of well-thought-out unifying ideas*, embodied in richer graphic expressions that are not chained to silly comparisons. These will be found by overall virtuality design (below), and not by metaphors, which I consider to be using old half-ideas as crutches.

Mistake 3: Add Ketchup

Some people think software will be improved by making it conversational and populating it with crypto-social entities—perky or sassy personalities full of greetings and apologies, that respond to, and in, some sort of English. (It is proposed in Apple's "Knowledge Navigator" film that they should even have animated faces, like Max Headroom.) I think it should be otherwise. We do not need gratuitous social interaction, but rather clear, sensible models of the working domain.

A *globe* is my model of a proper information system. A globe does not say "good morning"; it does not bother you with menus, icons, or prompts. You turn it and move your head to the most useful position for overview or detail, that's all. These crypto-social models have created a false trail of design directions, and are wasting effort and misdirecting hopes. Applying artificial intelligence to dandify bad designs is like putting on ketchup to dandify a bad meal.

Toward Virtuality Design

The Movie Analogy

It should be noted that the mistakes just listed all have to do with psychological and visual effects of screen events on the user. And in what field have the psychological and visual effects of screens been widely experimented with, and come to great prominence? Why, movies, of course.

Many people have now noticed that interactive software is in some ways like movies, and that the process of making software is in some interesting ways like movie-making.[1] Here are the central parts of the analogy:

- Making software is like making movies because both are about *how moving presentations affect the mind and feelings of the viewer.* The talented software designer subtly calculates the overall structure and how it will affect the viewer or user; not merely putting parts together. Technical concerns are merely preliminary, the substratum; what counts are the artistic planning, execution, and the reunified tuning of all the parts.
- The parts are subjected to a unified tuning process after they have been formally joined. We do not merely put them together, but trim, balance, time, subtract, and even reorganize them on new principles (see below).
- Special talents are required that have nothing to do with the technicalities. The greatest of these is the ability to conceive and realize a unifying vision.

I would like to consider software and movies both as parts of the same field, which I call *virtuality.*

---

[1] I first published this comparison in 1967 in an article called "Getting It Out of Our System" in Schechter, *Information Retrieval: A Critical View,* Thompson Books, 1967.

What Is Virtuality?

There are many ways of thinking about the responding computer screen. Perhaps it is just a technical device, the readout of a new form of calculator. Or perhaps it is just a new form of movie screen, where the lessons (and talents) of movie-making need to be applied.

I believe that movies and the computer screen are both best understood in still larger terms. It is for this I propose the term *virtuality* (as distinct from such terms as *artificial reality* and *virtual reality*, both oxymorons). The virtuality of a thing is what it *seems* to be, rather than its reality, the technical or physical underpinnings on which it rests.

Virtuality has two aspects: *conceptual structure*—the ideas of the thing—and *feel*—its qualitative and sensory particulars. In a movie, the conceptual structure consists of the plot and the characters, and the feel consists of atmosphere, suspense, and style. In architecture, the conceptual structure is the idea of a building and the sense of where things are, and the feel is the sweep and style and detailing. In a video game, the conceptual structure includes the rules and strategies, and the feel includes the tuning, spirit, motion characteristics, colors, and other sensory aspects.

The objectives of a designer will vary depending on the purposes of the virtuality. In almost all forms of design, we are concerned with conceptual structure: we want people to understand the result, and for it to be clear and useful to them. And in almost all forms of design, we are concerned with feel: we want the details of the finished work to be welcoming, pleasing, and easy for those who will see and use it.

Virtuality Design

The Design of Principles

In the metaphorical approach, the metaphor becomes the central concept—the principle, if you will, to which all other aspects of the design must adhere. The problem, as we have seen, is that slavish adherence to a metaphor prevents the emergence of *things that are genuinely new*.

What I call the design of principles is the more sweeping, serious alternative to metaphorics. In metaphorics, we begin with a familiar image or idea and try to pack into it, and tack onto it, some coherent set of functions that are constrained to be related to that metaphor. Freely designed *principles*, on the other hand, do not have to stay attached to any image. (This is why "metaphor" talk is counterproductive: it makes us try to stay within some clumsy concept such as "desktop metaphor" rather than reaching by the design of principles toward a new conceptual organization that has not previously existed.)

By a principle I mean an idea that other ideas are going to fit into or under. If you understand the principle clearly, the details follow from it.

Principles, unlike metaphorical correspondences, are plastic and redefinable. It is possible to fit ideas under the same set of principles in many different ways; conversely, the same ideas can be fit under many different principles.

The Jingling-Ideas Method

The design of principles consists of the detailed working-out of structures and ideas: we take a set of possible principles and endeavor to assign to them all the functions we want to include; then rework and rework, reassigning, redefining the principles and their visualizations until we get the best overall fit. The reciprocal nature of this process is the source of its strength: it avoids the unidirectionality of metaphorics (top-down) and featuritis (bottom-up) as design techniques.

I describe this process as the jingling-ideas method. When you hold a bunch of coins in your hand, they start as just a heap. However, if you shake them for a little while, in the right sort of way, they tend to assemble into a stack. This same method works in the process of design. If you think about all the things you want a design to do, and at first they don't fit together, the thing to do is *jingle them in your mind* until they stack up together. The result will be the gradual organization of these desiderata into a usable set of principles, aligning in some new fashion. Probably they won't include all the things you wanted, but you'll probably get most of what you wanted—in a clean design. (The more you can divorce your thinking from previous designs, the more likely you are to discover powerful new integrative principles.)

Combining the Old and the New

Today there are only a few organizing principles in interactive software: spreadsheets, databases, treating individual items as searchable, menus (in various styles, such as pull-down and Lotus), and windows, for example. Each of these involves an idea and a related visualization for it which together have a virtuality—conceptual structure and feel. Some of the principles derive from "old" (metaphorical) models; in the case of spreadsheets, it is the two-dimensional piece of paper; in the case of databases (including HyperCard), it is the three-dimensional stack of cards.

But some of the principles are also genuinely new. VisiCalc™, the first spreadsheet program, was a remarkable achievement: not only did it take the business concept of a paper spreadsheet and give it automatic calculation, but its whole structure was a tour de force in the design and balancing of principle. VisiCalc's REPLICATE feature is an excellent example of the abstract nature of the design of principle. To replicate a column *and its formulas* corresponds to nothing that was on earth previously; and when

metaphoric thinking was dismissed, it could be designed cleanly with no reference to anything that had come before.[2]

Tomorrow's new principles will involve both old concepts and new, fitting under new principles of visualization. These new principles must and will be *lucid*, *vivid*, and *obvious*, once you have seen them. They will be spatial because a screen is spatial, indicating multiple connection and multidimensional connection.

Representing Interconnected Information: An Example of Conceptual Structure Design

Once we leave behind "two-dimensionality" (virtual paper) and even "three-dimensionality" (virtual stacks), we step off the edge into another world, into the representation of *the true structure and interconnectedness of information*. To represent this true structure, we need to indicate multidimensional connection and multiple connections between entities.

We can think of the new connective structures as being basically of two different types.

- *Multidimensional connections*: Uniform or systematic spaces. The data on a globe is a nice example of a systematic space.
- *Discrete connectedness*: Mapping individual and disparate interconnections, as in hypertext, hypermedia, etc.

Tomorrow's file servers, such as the Xanadu™ electronic storage and publishing system, will maintain various types of connections among files or documents. But that is just the technical mechanism for holding the connections. How will we actually see and use them? Working on this problem has led to the emergence of two new principles. The principle of *connected windows* combines two "old" principles to form a new one. The principle of *wormholes* (out of which information may be enticed) is a "new" principle created to provide a visualization for unseen parts, continuities from the seen to the unseen.[3]

Designing the Feel and Fine-Tuning

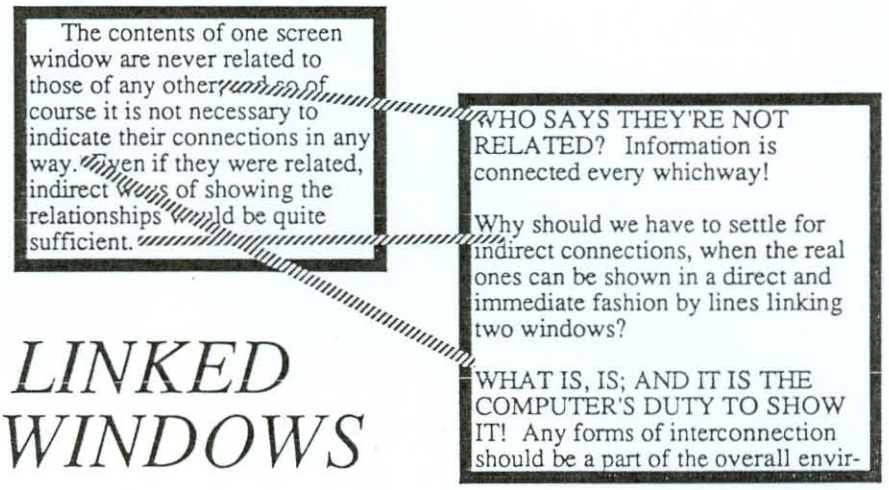
Virtuality is to some extent designed as a whole (as in the movie script), to some extent evolves as the parts are made, and to some extent is fine-tuned (as on the movie set and in the cutting and mix).

Consideration of feel comes generally, if ever, after the conceptual structure. Sometimes it is something that can actually be carefully planned and

---

[2] For a fuller discussion, see pp. DM 69–71 in T. Nelson, *Computer Lib*, 2nd edition, Microsoft Press, 1987.

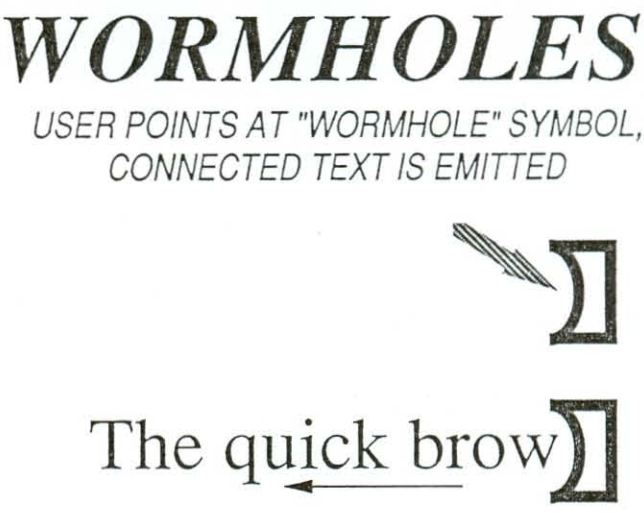
[3] Some children's books published earlier in the century had illustrations with die-cut doors you could look behind; so did *Flair* magazine, a publication of the late 1940s. These should be an inspiration.



engineered, like the handling of a car (which in today's automotive world involves calculation as to steering, the distribution of weight, shock absorption, the angles of struts, etc.). But the feel more usually comes in the rework and fine tuning.

Beginning with the overall virtuality that is wanted—the conceptual structure and feel—the designer then works back to design the particulars. If it is a movie, the director takes the script down into the details of each scene and atmospheric nuance. If it is a program, the designer works on the particulars of every feature and tries to configure each so that it fits seamlessly into the whole.

When most of the separate parts have been completed, the tuning phase begins. Like film editing, the process consists of intercomparison decisions about fragments already in hand. In the tuning phase, we work to get the feel just right (and usually modify the conceptual structure to a lesser extent,



sometimes throwing parts out and sometimes assimilating them to new or revised generalities).

On Artistry

Historical accident has kept programmers in control of a field in which most of them have no aptitude: the artistic integration of the mechanisms they work with. It is nice that engineers and programmers and software executives have found a new form of creativity in which to find a sense of personal fulfillment. It is just unfortunate that they have to inflict the results on users.

Learning to program has no more to do with designing interactive software than learning to touch-type has to do with writing poetry. The design of interactivity is scarcely taught in programming school. What we need in software is what people are taught in film school, at least to whatever degree it can be taught. Designing for the little screen on the desktop has the most in common with designing for the Big Screen (directing theatrical films). Interactive software needs the talents of a Disney, a Griffith, a Welles, a Hitchcock, a Capra, a Bob Abel. The integration of software cannot be achieved by committee, where everyone has to put in their own addition (featuritis again). It must be controlled by dictatorial artists with full say on the final cut.

As computer hardware improves, tomorrow's production values should go up and up. Tomorrow's systems will be high-performance, full-flavored, profoundly rich. The difference between tomorrow's software and today's will be, to use Mark Twain's telling phrase, the difference between the lightning and the lightning bug.