Intuition versus ease of use

Ben Klemens

15 May 2009

[Part three of six]

A book entitled *Design of Everyday Things*, by Donald A Norman [Norman, 1988] very clearly had an influence on the design of many of Microsoft's products. It in turn was influenced by what was trendy at the time (1988): the original Macintosh features prominently, there is a whole page on the promise of hypertext, and he complains about EMACS. In his section on Two Modes of Computer Usage, he explains that there's a third-person mode wherein you give commands to the computer, and then the computer executes them; and there's a first-person mode where you do things your own darn self, like telling the computer to multiply matrix A by matrix B versus entering numbers into the cells of a spreadsheet. At the ideal, you can't tell that you're using a computer; the intermediary dissolves away and it just feels like working on a problem. Of course, some tasks are too hard for first-person execution, as Mr. Norman explains: "I find that I often need first-person systems for which there is a backup intermediary, ready to take over when asked, available for advice when needed." This paragraph, I posit without a shred of proof, is the genesis of Clippy the Office Assistant.

Although Mr. Norman points out that we feel more human and less like computer users when we are in first-person mode, it is often a terribly inefficient way to work. A word-processor document is *not* like handwriting a letter, so pretending it is is sometimes folly. For example, you don't hard-code numbers: instead of writing Chapter 3, you'd write Chapter \ref{more_rambling} (LaTeX form; Word has a similar thing), and let the computer work out what number goes with the more_rambling reference.[1]

In the context above, first-person mode matches literal markup. Don't write a note to the computer that it should find all titles and boldface them; instead, go and boldface them all the way you would if you had a highlighter and paper in hand. Third-person commands are inhuman, unintuitive, and how we get computers to make our lives easier and more efficient.

Forcing the user DOET has much to say about saving the user from him, her, or itself. Make it impossible to make errors, he advises designers. His shining example of good design are car doors that can only be locked from the outside using the key. There's a trade-off of some inconvenience, but it is absolutely impossible to lock the

[1]I used the LaTeX markup for Chapter \ref{more_rambling} here because it saves me the trouble of having to explain the seven-step process it takes to do the same thing in Word. And by the way, if you change the referred-to chapter's title, all of the references will break and you'll have to repeat the seven-step process for each reference.

car keys inside. Word clearly fails on this one: you want to hard-code your references? Feel free; in fact, we'll make it hard for you to do otherwise, since doing otherwise doesn't follow the metaphor of simply writing on paper.

More generally, a good design has restrictions: if you can only put your hand in one place on the door's surface, then that's where you'll put your hand, and the door will open on the first try. What about LaTeX? It gives you a blank page. You can type a basically infinite range of possibilities. This is where DOET leaves the command line: it isn't restrictive enough to guide the user, and therefore is a bad design.

I think he's got the interpretation entirely wrong: there is only one thing that you can do with the blank slate that you get in EMACS, LaTeX, or a command line: read the manual (RTFM). Just as your car won't let you lock yourself out, you can't write a crappy document in LaTeX until you've gotten a copy of the manual and at least had half a chance to expose yourself to the correct way to do things. Mr. Norman again: "Alas, even the best manuals cannot be counted on; many users do not read them. Obviously it is wrong to expect to operate complex devices without instruction of some sort, but the designers of complex devices have to deal with human nature as it is." True, people won't read manuals unless you force them to. So force them to.

Ease of initial use The benefit of the intuitive interface is that you don't have to read the manual.[2] You can jump in and go. Aunt Myrtle only writes one letter a month, so making her spend an hour reading the introduction manual—which she will entirely forget by next month—is inefficient and bad design.

But ease of initial use is only important for those items that we only use once or occasionally. Think of the things you use every day: your preferred means of transport may be an automobile, a bicycle, or your shoelaces. You spend all day typing with a QWERTY keyboard. Perhaps you play a musical instrument. The fact that you are reading this indicates that you are literate. None of these things are intuitive. You spent time (in some cases, years) learning how to do them, and now that you did, you enjoy driving, riding, playing, and reading without thinking about the time you spent practicing.

Simply put, not having to read the manual is massively overrated. If a person is going to use a device for several hours every day for the next year or even the next decade, then for them to spend an hour, and maybe even weeks, learning to use the device efficiently makes complete sense. More on this important point later.

Metaphor shear Another problem is what Neal Stephenson calls *metaphor shear*. That's when you're happily working with a mental model in the back of your mind, and one day your metaphor breaks. Back to DOET: "Three different aspects of mental models must be distinguished: the *design model*, the *user's model*, and the *system image* [. . .]. The design model is the conceptualization that the designer had in mind. The user's model is what the user develops to explain the operation of the system. Ideally, the user's model and the design model are equivalent. However, the user and designer

[2]By the way, I rarely find intuitive interfaces to *actually* be intuitive. They're designed around certain target users whom I'm evidently incapable of thinking like. More generally, the concept of having an intuitive interface assumes that the intuition of everybody on Earth is exactly the same.

communicate only through the system itself: its physical appearance, its operation, the way it responds, and the manuals and instructions that accompany it. Thus, the *system image* is critical; the designer must ensure that everything about the product is consistent with and exemplifies the operation of the proper conceptual model."

This is where DOET overestimates computing. It's a book that's mostly about doors and faucets and other everyday objects. He's right that if you have to RTFM to work a door (even if the manual just says *Push*), the door's design is broken. He's right that for complex systems, like panels of airline instruments, they should not work *against* intuition (e.g., if two levers do different things, they should look different). But he combines them into a false conclusion: complex systems should work with intuition so well that you shouldn't have to read the manual.

First, this is absurd in any setting but desktop computers. Would you feel OK if your pilot told you the plane was so intuitive that she didn't bother learning how to use it before the flight?

But back to the main point, making a word processor which is so intuitive to the user that he or she doesn't have to RTFM is a *much* more complex task than making a manual-less faucet. If we needed to build a faucet such that it runs if the user presses it with his hand, bangs it with a pot, or bumps it with his elbow, that would be easy—put a button on the top. But to program a picture of a faucet such that the user can click on the thing, or double-click on the thing, or type R and all make the picture of a faucet run requires programming a call to the Run method for three separate events. If the user comes up with something that the programmer didn't think of, like holding down the alt key and clicking on the picture, then the user's metaphor shears. What your momma told you is true: it's easier to just present the truth than to weave a whole world around a lie.[3]

Mr. Norman's call for simple interfaces (he doesn't really say anything about metaphors to physical objects, but he does talk about simple mental models, and for most of us that means physical metaphors) therefore leads us down a supremely difficult path: first, the program designer must lie to the user by presenting a metaphor that is easy for the user to immediately guess at. Then, the designer must now design the program so that anything the user does, no matter how unpredictable, will cause the program to behave in the correct metaphorical manner. This is a very high bar, to the point that a program as complex as Word simply can not achieve it.

Feature creep Mr. Norman is right that we shouldn't have to RTFM for simple, everyday tasks. Writing a letter or one-page paper is so common that his principle that it should be manual-less should probably apply. Further, we have the technology. However, as I've learned ever-so-painfully, writing a book is an order of magnitude more technically difficult. Programs like Word and Scientific Word imply that writing a letter and a book are are identical, just a matter of extent, when in the end they aren't: one has a valid paper metaphor attached, which programmers can easily implement,

[3]For those down with the lingo: every event has to have a method for every object, which is dozens of events times dozens of objects equals hundreds of things that could go wrong with the metaphor—assuming you got good rules about passing the right events to the right objects to begin with. Inheritance doesn't help because most of the time the inherited methods don't quite work as they should, leaving you with objects which almost fit the metaphor.

and one does not. A good word processor, then, would let you do basic things without effort, and then put its foot down at some point. You get all the tools you need to write a business letter, and then if you want more, you'll need to get a new tool with a manual. Clearly, nobody is ever going to write a program like this. To some extent, this is a good thing, since it pushes technology forward, but at the expense of annoying users who have to sit through half-appropriate metaphors badly implemented. Mr. Norman writes about creeping featurism as an evil which pervades all of design, and he's right: nobody ever says "I'm done."[4]

One good way to implement this would be a simple graphical front end to the basic features of a less metaphor-laden back-end program. When you've sapped the offerings of the graphical front end, you'll have a bearing when you RTFM on the less intuitive stuff. This is how a host of Unixy programs work, but the front ends also eventually succumb to featurism. Scientific Word takes it to the extreme, by trying to give you a button for every last feature and refusing to admit that it is a front-end—perhaps because it is an expensive front-end to free software.

Since no programmer will ever have the discipline to admit that their manual-less tool will work only for a limited range of tasks, the discipline falls upon the user to realize that it's OK to use simplifying metaphors for simple situations, but complex tasks require tools that don't lie to you.

Word is carefully built from the ground up to be intuitive, not to be efficient—and it lies to you every step of the way to give the impression that the system actually works the way you intuitively guess it does. The next section describes how even the smallest intuitive but inefficient detail can add up to immense time costs in a system you use all day, every day.

References

Donald A Norman. *The Design of Everyday Things*. Basic Books, 1988.

[4]There is a stand-out exception to this: TeX was done in 1988, after nobody claimed the author's cash prize for finding bugs, and the code base has not changed since then. The add-on, LaTeX, was cemented in 1994. Authors who want to change something in the system must add a package to the base systems.