Jef Raskin
8 Gypsy Hill
Pacifica CA 94044
415-359-8588 415-359-9767 Fax
KE6IGI—jefraskin@aol.com—AMA L-88

February 15, 1997

THE SWYFT AND THE CAT

note: all company, feature, and product names are trademarks or registered trademarks.

A number of interface directions that are only now coming into prominence were foreshadowed by a series of products from Information Appliance Inc. (IAI) None of the products achieved wide recognition at the time, but interest in their revolutionary interfaces has steadily grown. This article was written in response to many requests for information about this work. The best known of the products was the Cat, a desktop machine designed for the well-known Japanese firm, Canon Inc. Information Appliance's SwyftWare for the Apple II series and the Swyft portable computer were the other major designs that came from IAI. The chief architect of these products was the company founder. He also originated the Macintosh project at Apple computer. The aim of this essay is not so much to present a retrospective as to inspire.

[Photo of the Cat]

In many ways subsequent work, such as Gelertner's LifeStream chronological and linear file model is not very different from the methods incorporated in the effective and simple SwyftCard product for the Apple II, introduced in 1983. The flavor of document-centered paradigms are currently hinted at in software such as Apple's OpenDoc and Microsoft's OLE, but a much fuller expression of docucentric computing permeated all of Information Appliance's products.

According to information from Canon, Inc., about 20,000 Cats were sold. While the Cat was in fact a bit-mapped computer with a Motorola 68000 microprocessor, with the same screen size and inherent power as contemporary Macintosh computers from Apple, Canon chose to sell the Cat though their electronic typewriter division, and limited the use of graphics to those that could be reproduced by the daisy-wheel printer that they sold with the it--namely none. Nor did they realize the importance of third-party software though Information Appliance had begun to get third-party development started (especially in the file-conversion area). Though most customers never knew of or benefitted from the Cats full capabilities, the reaction of users and reviewers alike to the product's ease of learning and use was overwhelmingly positive. [look up and quote some reviews].

THE SWYFT

The Swyft was a small portable computer with built-in software for many common computer-

based tasks, such as word processing, graphics, data-base operations, spreadsheet, and telecommunications. The user could easily create macros to automate often-repeated tasks. To encourage software development, it was its own programming environment, though the key to enable programming was not widely disseminated as most users have neither the desire nor the competence to do software development. Standard telephone-style jacks were provided for telephone communications, there were both serial and parallel ports, the serial port being Mac-compatible, the parallel IBM-compatible.

[Photo of the Swyft]

The specifications for the Model III Sywft are: weight 4.0 lbs (1.8 kg), 11.8" X 10.6" X 1.3" (300 mm X 270 mm X 32 mm), 8 AA Nicad batteries were good for 6 hours of use (longer with Alkaline batteries), 640 X 200 supertwist Liquid Crystal Display, 13 function keys, a 2400 baud modem, and 512K of memory. The software handled file transfers to and from both Macs and IBM compatible computers. The floppy drive was external and the retail price of the Swyft Model III was to be $999. The Model I with less memory and no modem was $799. 2400 baud was fast at the time, and the memory size was very large for a portable.

One of the fundamental principles of common-sense interface design is that simple things should be simple. A single button turned Swyft on, there was no delay. If you wanted to write something, you just started typing. You needed no commands, you just start typing. All text was inserted, you could not wipe out text by typing over it.

The principle that a user's work is sacred, and should not be changed without a user command or as a side effect of the action of some other command, is an important one. Its violation can be costly. Cut-and-paste paradigms are problematical in this regard and deserve some comment, especially as we did not learn the lesson well enough when designing an analogous feature in the Cat.

CUT-AND-PASTE AND OTHER DESIGN ERRORS

In almost all of today's word processors you can replace text by highlighting it and then typing. The highlighted text disappears and your typing replaces it. This design feature was intended to save the step of having to explicitly delete the highlighted text, that is instead of select-delete-type you need only select-type. Deletion is a byproduct of typing when there is a selection. When I gave a talk at the Bay Area Chapter of the Association for Computing Machinery's Special Interest Group in Computer-Human Interaction (Palo Alto Ca 11 Feb. 1997) I asked how many people had never lost any text due to this design. Out of the some 300 people there, only one person raised her hand.

What makes this design flaw especially grotesque is that the selection may not even be visible on the display, and it may be of any size. You can be looking at a display, not noticing that you have forgotten to or failed to move the cursor (which is not always particularly conspicuous) to the current screen and start typing. The display suddenly shifts and you can see that you are typing on some other screenful, but you do not see that you have just deleted an arbitrary amount of

text.

When designing the Cat, we made a similar design error. Originally, to move text, we had the user select the text to be moved, place the cursor where it was to be moved to, and then tap a special key to invoke the MOVE command. This way there was never that uncomfortable interim period found in CUT-and-PASTE-based moves where you have just deleted the text but have yet to place it anywhere, and a subsequent CUT costs you the text you cut while intending the first move. This is another design flaw that occasionally causes lost text for most users.

Noted author and designer Scott Kim, then working for Information Appliance, pointed out that we could eliminate the MOVE command by having the rule that if the cursor was moved (via the LEAP command, which will be discussed later), any highlighted text would automatically be moved. The sequence of actions became select-and-then-just-move-the-cursor. The text followed the cursor and stayed selected so that you could move it again if the move had been to the wrong place. The UNDO provided additional safety. Moving text this way was a safe shortcut as in the worst case, we argued, since while you might accidentally end up with some text where you didn't want it, at least you'd never lose anything. You could always move it back. This seemed wise counsel, and we adopted this design for the Cat.

In practice it was not too bad, most of the time. But making moving text a side-effect of cursor motion turned out to not be entirely successful, precisely because it was a side-effect that you might not be paying attention to when your attention was on the act of moving the cursor. I can remember vividly a few particularly annoying cases that happened to me. In one case I had selected nearly half of a fifty-page essay, for some reason. All of a sudden I remembered a phrase that I had wanted to further explain. I moved the cursor to that phrase and typed away at my idea with full speed. Of course, a huge chunk of my essay got pulled into the wrong place and my subsequent typing and editing, made without observing that I had moved a mass of text, eliminated all possibility of using the single-level UNDO.

I continued writing and editing, and only much later noticed that something was wrong. It is amazing how hard and frustrating it was to figure out what had happened! I felt that the text had been scrambled. And once I had figured out the problem, it took a lot of searching and reading to further discover just what had been moved and where, exactly, it had been moved from. Another occasion was when a few digits got accidentally moved in some data. It was impossible to figure out what had been moved as there was no redundancy.

Other users reported the same occasional difficulty, but due to a psychological phenomenon that Don Norman has so often pointed out [reference] they blamed themselves for misuse of the product rather than the design flaw that I had approved.

TWO CURES FOR CUT-AND-PASTE PROBLEMS

In the Swyft we returned to the Cut-and-Paste model, but added a new feature where all cuts were stored sequentially in a system-created document (a separate "permanent delete" command allowed the permanent erasure of text from that document). This prevented accidental loss of text

(due to the causes discussed in this section) and the collection of cut material often proved a valuable resource. The first item in the document is what would be pasted when the PASTE command was executed. Deleting something from the Cut Document merely moved it to the beginning.

It would be a useful study to compare cut-and-paste-with-store as we did for the Swyft with the original MOVE command, with the alternative of having each document have its own associated cut document, and possibly some other alternatives, to see which causes the fewest disasters. It would not be an easy study, as the frequency with which the problem occurs is low. But low-frequency high-impact design errors are just the kind of thing that makes people edgy about and dislike a product. As various experiments have shown [cite some], an occasional, seemingly random punishment for what is normally a permissible or even desirable act can cause neurotic behavior in animals.

We should design so that this kind of punishment is never dealt us by software.

Neither the Cat nor the Swyft, fortunately, allowed deletion of text to be a byproduct of typing as do almost all current word processors, one of the reasons they were singularly unfrustrating.