INTRODUCTION

When computers were new, nobody had fun with them except, possibly, the people who created them. The computers themselves were locked away in special rooms and not everybody had access to them. Users, even the most serious of programmers, spent many hours going over their programs and other input just to be sure it was perfect. The act of programming was carried out at desks, using paper and pencils.

When the programmer was finished, another group of people translated the program into a set of punched cards. This was a particularly slippery and risk-prone embodiment of the hours of work the programmer had already put in. When the card deck was ready, the programmer or someone else took the stack of cards to an input clerk.

The input clerk had tremendous power. She (they were mostly women) decided whose jobs could jump ahead in the line. Hours later, the programmer got back his output, generally in the form of a printout. If everything went well and there were no mistakes of form or logic, the results would be useful. If either the programmer or the keypuncher made even one tiny slip, all the hours of work and waiting went to waste. Even if the mistake was a trivial or easily discovered one, the programmer had to wait for his next turn to have his program run. In many installations, programmers got only two or three runs per day.

The whole system was geared to make the computer important and the user unimportant. In fact, users in the current sense did not exist. Essentially, everybody who used a computer was a programmer or a keypunch operator or some other kind of specialist. The computer's time was considered valuable, so valuable that it was measured in
expensive seconds. The programmer could work for hours in order to save the computer a few seconds.

There was a great separation between the computer and the users, both physical and psychological. The programmer worked only at a distance with the material of the computer world. Except for a few visionaries like Vannevar Bush, nobody foresaw today’s highly interactive computer world.

Nevertheless, the rewards of doing something new and creative, and the thrill of getting the computer to do something were there. People like Grace Hopper had plenty of fun, and so did the other computer creators. Within a few years, interactive computing began to flourish, and the creation of software that posited an active user was common. Still, the users had to do what the programmer wanted them to do, within the limits set by the programmer. Clearly, someone else was in control.

The subject of this paper is the way that users get to share what programmers have—the feeling of control over and comfort with their machines. In the dozen years of the personal computer’s existence, the role of the user has moved from grateful but miserable wretch or computer whiz to kingpin. Several types of user-modifiable interfaces will be discussed, including: keyboard redefinition and macro programs; macro facilities built into spreadsheet and word-processing programs; “work” menus created by the user; full-blown customizable interfaces; and Apple’s HyperCard program for the Macintosh. How each kind of user-controlled interface empowers the user and changes his or her relation to the computer will be discussed.

The existence of and the acceptance by users of all these interface control tools both enabled and marked a fundamental change in the role of the user. Users have become more like programmers while still remaining users. Without being highly technical, without thinking of themselves as “computer whizzes,” people who use computers for productive daily work as well as pleasure have come to feel that they are in control of the computer. By acknowledging user need to control the interface, programmers have divested themselves of some of their specialness and shared some of their satisfactions with users.

Eventually, computers became more common, and more work took place at terminals, first printing terminals and, eventually, video terminals. The rise of video terminals drew the user closer to the computer in two ways. First, it made the programming process more interactive and immediate. No one mediated the programmer’s contact with the computer. Second, it made interface important. Programmers had to use keys and screen displays to get anything to happen.

Video terminals made possible interactive programs intended for
ordinary users rather than programmers. People who did not want to devote their lives to computers began to use computers for tasks ranging from data entry to financial modelling. A separate class of computer-like machines, dedicated word processors, came into being. These had no programmers, only users whose goal in using the machine had nothing to do with the computer itself. However, even dedicated word processors were run by key operators, who had to know far more about the operations of the machine than the rest of the users did.

What makes computers attractive to programmers? What do programmers like about programming? These questions are related to ordinary dedicated programmers who bring some passion to their work. The discussion is based on the author’s own experience as a programmer, conversations with other programmers and users, and from such accounts of programmers’ experience as Sherry Turkle’s The Second Self (1984). A strong common thread is the experience of control. For programmers, the computer is a place they can control and understand. The computer does what they want it to do, the way they want it to. Of course, this is an ideal, and it takes plenty of work to get the computer to do the right thing. But the programmer is in control. He or she determines what the computer will do, and the computer always does what the programmer tells it to. The problem is that only the computer knows what it has actually been told to do. Hence, the frustration of debugging.

Closely related to control is the urge to personalize the computer. If the computer is the world the programmer interacts with, the programmer wants to make it his or her world. There are plenty of equivalents in the computer world of those paintings one sees on the sides of vans. Programmers can determine the wording of the prompt they see. They add their own twists to the operating system and the editor they use. Many of the small twists they add have about the same function as putting up pictures on the door of an anonymous dormitory room—they show who is the owner.

Programmers also share an urge to fiddle and play. Lots of programmers also have hobbies like ham radio and model railroads—the kinds of toys that permit and reward endless fiddling. If one doesn’t want to work but still wants to be in contact with the computer, there are plenty of housekeeping things to do. The IBM-PC owner can alphabetize files in directories, and the Macintosh owner can move files around between folders, and it looks like work. For people who control a larger system, the scope for play is much larger. One of the best accounts of this was written by Ray Ozzie (1986), formerly a PLATO system programmer who developed the Symphony program for Lotus. His almost poetic description of the fun to be had with the PLATO system when the users were all home in bed can be found in a book
called *Programmers At Work* (Lammers, 1986), a hymn to creativity and fun that makes inspiring reading.

Another thing that programmers like is the feeling of immediate gratification. Despite the problems involved in debugging, when the program works, it works. The rewards are small and constant. Frustration only increases the intensity of the reward when the program finally works.

The rise of the personal computer gave programmers the ideal field in which to create interactive programs. Interactive here means programs that are built on constant communication between the user and the machine. The first personal computers required the utmost in understanding from their users. One had to be not only a programmer but an electronics expert to get anything out of them. They came in kit form and had to be assembled and then programmed by flipping switches on the outside of the case. Even when BASIC came along and people could program in a traditional manner, there were no plain users. Everyone was a programmer. Ease of use was not a goal.

The first few commercial programs and games hardly altered the situation. Only when VisiCalc gave ordinary people a reason to use the computer did a large class of nonprogramming users arise. These pioneers experienced a lot of difficulty. Computers were still tough to use and computer programs still rigid and mysterious. People joined user groups in order to get enough information to use the machines properly, and to share their computer frustration.

When the personal computer became more commonly used for productivity, the class of pure users arose. These users did not see the computer as a challenge to their skill and understanding. They did not want to play with their computers. They wanted to use the computer as a kind of typewriter/math machine/toaster. So interfaces were made easier for them. Programmers and designers began to take the idea of ease of use seriously, seeing that the audience was no longer their fellow hobbyists.

After a few more years, the sophistication of the users rose, and they began to see the computer as something they could control. Enough experience with computers gave users sufficient understanding of the computer to imagine better ways of working with it. The availability of interface-altering tools brought the users’ concept of what they did with a computer much closer to the programmer’s idea. Without having to do programming-surgery on the programs, users could make programs behave the way they wanted them to. Users could master the computer world they lived in without having to become experts. A new kind of equality arose between users and programmers.
Several types of user-modifiable interface tools have arisen over time. Some of these are:

- keyboard redefinition and macro programs (such as Prokey, Keyworks, Superkey, Tempo, and Quickeys);

- macro facilities built into spreadsheet and word-processing programs;

- "work" menus where the user can promote any item buried in the regular menus to a special top-line menu;

- full-blown customizable interfaces as found in programs such as Borland's Quattro and Sprint, where the user can create his or her own set of menus using an interface-creation language not unlike a macro language; and

- Apple's Hypercard program for the Macintosh, a program that consists almost entirely of interface.

This list includes only methods where the user can control the interface within a program. Another large but amorphous class of user-controlled software customizes the computer system itself. Into this class fall the hundreds of utility programs that can alter the directories searched by the computer or allow the user to review what just scrolled by; or blank the computer screen after a period of inactivity; or make the irritating, blinking cursor into a friendly, steady, solid block; or supplement the Macintosh Finder with a more traditional method of selecting, moving, and deleting files. These are not discussed because they are less user-oriented. They allow the kinds of personalization that programmers enjoy, and some of them make life much easier for those who employ them. But they don't empower the user in the same way as the other tools being discussed here.

**KEYBOARD REDEFINITION AND MACRO PROGRAMS**

Keyboard redefinition and macro programs came to the personal computer in late 1983. Well-known programs of this type include Keyworks, Prokey, and Superkey for the IBM PC; and Tempo and Quickeys for the Macintosh. These programs use a single mechanism to do several things. They allow the user to decide what will happen when a given key is pressed. One can decide that when one presses,
say, CTRL-Y, the computer will hear, instead, CTRL-Z. If a familiar program does something harmless like open up a dictionary when CTRL-Y is pressed, but a new program deletes the entire page on the same keypress, it is in the user’s interest to block off the now disastrous CTRL-Y. So users made programs mimic each other, and protected themselves from destructive mistakes.

If accented letters or other special characters that require horrendous finger twists are regularly needed, the keyboard can be redefined so that some unused combination gives the needed letter. In other words, the user decides what is important enough to be accessible. Users can also assign whole strings of keypresses to a single keypress. A single keystroke could issue the commands to, say, type a standard letter closing and signature, open a new worksheet and fill in standard headings, or close one program and open another. In fact, keyboard macros can automate all kinds of procedures. Users can essentially add new features to a program by assembling operations and putting them on a single key. This is tantamount to inventing new products. In some cases, users are remaking the world—for example, deciding that they live in the kind of rational world where the Dvorak keyboard layout won out.

Some macro programs now have additional capabilities for defining menus. The menus can consist of operations already included in the product or of concatenated operations such as showing the list of allowable entries for a field, letting the user choose one, and typing that entry. Again, a totally new feature is added to a program’s interface by the user.

The abilities to make new programs act like old programs, to decide what features should be easily accessible, to automate work, and to add features to a program certainly do add to productivity. But they also make users feel in control. The users are deciding how the program should work. They have escaped the control of the programmer and designer.

**Macro Facilities Inside Programs**

Most major productivity programs, mainly word processors and spreadsheets, include the capability for creating macros. These differ from the separate keyboard macro programs in two ways. First, they often include ways of addressing the capabilities of the program they are designed for, beyond simulating the pressing of keys. Second, they can be created in two ways: either by direct construction in a kind of “macro language” not unlike a programming language, or by “watch me” where the computer records every action the user takes and records it as a macro. The user need only assign the resulting macro to a
keyboard, and the computer will do it at any time. This opens the creation of macros to users who cannot or do not care to analyze operations and create commands to carry them out.

Built-in macros have become the hallmark of the sophisticated product. The first incarnation of Lotus JAZZ was panned because it had no macros. Even though it was intended for novice users, it was judged harshly because of the lack of macros.

Sophisticated users make entire applications out of macros, hiding the product’s original interface and creating menus or controlled data-entry forms usable by anyone, often by someone much more naïve than the macro creator. Here, users actually take on the programmer’s role, creating a piece of computer interactivity for someone else to use.

“Work” Menus

Some new products, especially Microsoft Word for the Macintosh, include a menu named “Work.” Users can add items from other menus to the Work menu, along with documents and combinations of menu actions not unlike macros. What this does is make what the user wants accessible, accessible. The user, not the programmer or designer, decides what is important enough to be on top. In yet another way, the product can be suited to the user’s manner of working.

A lot of these changes just are not that important. While they add convenience, they are not indispensable. Their main function is to put the users in control. Unfortunately, people get dependent on their customized version of the computer, especially with the operating system utilities not discussed in this paper. If one is used to a computer whose keyboard had been redefined, and moves to a machine which does not include that set of keys, increasing personalization does have a backlash.

User-Defined Interface

A few brand-new products, most notably Borland’s Quattro spreadsheet, allow the users to define the interface completely, without using macros. Quattro users have three choices: they can accept the Borland set of menu trees, use a menu tree designed to imitate Lotus 1-2-3 closely, or invent a new menu tree. A fourth menu tree is available, designed to simplify things for novices. However, the tasks for novices are different in each office. An advanced user in the same office will invent situation-specific menu trees that make the tasks of that office easier.

Menu trees are created using essentially the same language as is used for macros. Each menu operation and keystroke has a name, and the menus are assembled from names selected by the user. A user can
start from any of the three menu trees provided and make small alterations, or start from scratch and take on the whole design task.

The complete user-defined interface does everything that the other user-modification tools do, and more. Users can make an unfamiliar product imitate a familiar one, or add functionality, or decide what is important enough to put at the top level, or what is similar enough to belong together. Users can design their own product without really programming.

But something else is happening, in a purer form than with the other tools. Users who create a menu tree must not only imagine a slightly better way of doing things, must not only long to automate something they already do, but also must analyze what they really do, and how they use a computer. They also must imagine how they would use the computer if they could. Introspection, analysis of their own learning style and working pattern, and a way of imagining the computer as not fixed but their own, all enter into this process. Users become not programmers but either cognitive scientists or teachers. These are probably more comfortable roles because, even though one has to use the analytical skills of a programmer to build an interface using these tools, one does not have to think of oneself as a programmer.

Borland has been advertising a new word processing program called Sprint which also has a user-definable interface. It has been seen by a few people and imagined by a lot more. Word processing is probably the most personal of computer applications, since it is supposed to be invisible. People have strong ideas about what they want in a word processor. They also have passionate attachments to what they are used to. Many offices want to share files but cannot do it easily because everybody is bonded to a different word processor. Maverick employees use their own favorite, and waste lots of time converting their work when the times come to share it. With Sprint, the barrier breaks down immediately. Sprint will come with interfaces emulating the most popular word processors, and probably with a few new interfaces. So the comfort level will increase for people who already love their own word processor, and people who know they can make a better mousetrap are free to try. Control over the computer world is almost complete. The remaining problems are mostly hardware problems, i.e., computers can still break, of course, and they are always too slow.

**Hypercard**

Hypercard is a program that is all interface. It exists to be an interface to information. Hypercard programs come in the form of stacks, which can be a little bit like databases or a little bit like interactive
WHEN USERS CONTROL THE INTERFACE

education, or a lot like front ends to complex information. With Hypercard, the user is always looking at a screen covered with pictures, information, and potential actions. The user takes an action, usually by clicking the mouse someplace, and the stack processes that action.

A major use of Hypercard is to organize information. One can make phone lists, daily appointment logs, or a customized periodic table chart. Hypercard is sold with some generally useful stacks, including an address book and a calendar, which are meant to be altered to suit the user.

Hypercard invites and even demands customization. Even if one uses only the ready-made stacks, additions are needed to make them useful. And changing relatively simple things is incredibly easy. Of course, doing complex things is hard, but the ratio is reasonable—the amount of effort it takes to do something complex is more or less commensurate with the complexity of the task. This is in stark contrast with some traditional programming and macro languages, where simple operations are nearly as difficult to set up as complex ones. In fact, some operations that are ferociously complex to program in many languages are simple in Hypercard. And the complex tasks are done as extensions of simple tasks. Easy tasks should be easy and hard tasks should be possible—that’s the hallmark of a useful and flexible tool.

Hypercard makes it so easy to modify programs to one’s taste that people are once again tempted to fiddle with their computers in the way that BASIC once made the original personal computer users do. The level of excitement generated by Hypercard among users (as opposed to people who expect to profit from it) is extraordinary. This excitement is happening because Hypercard lets people use their computers the way they always imagined—more like a very smart typewriter and less like HAL.

CONCLUSION

One does not have to love the computer to be an effective user. But the ability to control the computer reduces the fear. All these tools that are now available make it possible and attractive for mere users to control and personalize their computers. The gulf between user and programmer has been reduced. The growth of the nonprogramming power user both made possible and was made possible by the rise of user-interface tools.
REFERENCES