1

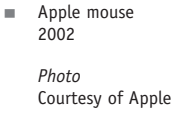
The Mouse and the Desktop

Interviews with Doug Engelbart, Stu Card, Tim Mott, and Larry Tesler

When you were interacting considerably with the screen, you needed some sort of device to select objects on the screen, to tell the computer that you wanted to do something with them.

Douglas C. Engelbart, 2003, referring to 1964

Why a Mouse?

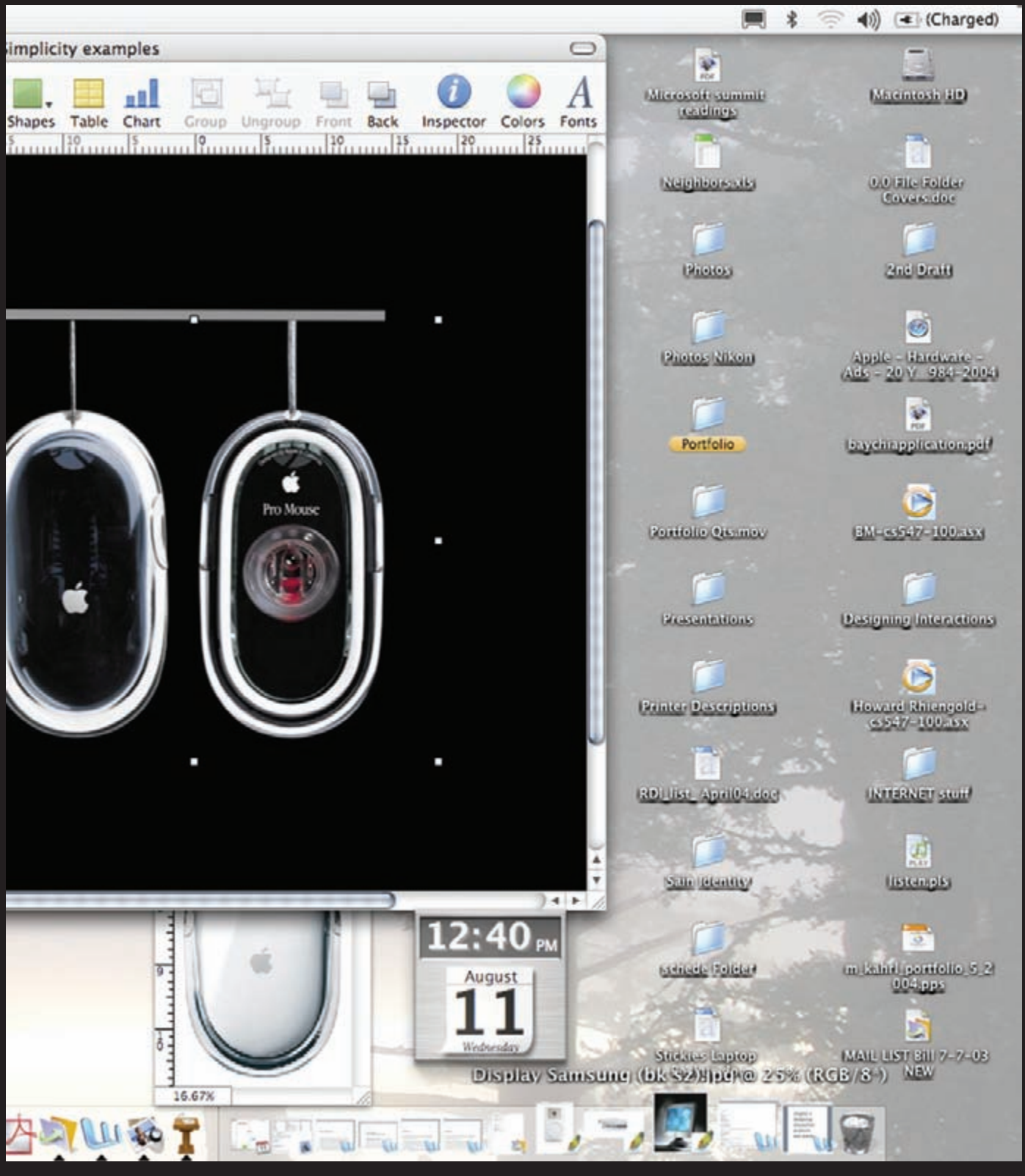


Apple mouse
2002

Photo
Courtesy of Apple

WHO WOULD CHOOSE to point, steer, and draw with a blob of plastic as big and clumsy as a bar of soap? We spent all those years learning to write and draw with pencils, pens, and brushes. Sharpen the pencil to a fine point and you can create an image with the most delicate shapes and write in the tiniest letters; it's not so easy to do that with a mouse.

Doug Engelbart[1] tells the story of how he invented the mouse. When he was a student, he was measuring the area under some complex-shaped curves, using a device with wheels that would roll in one direction and slide sideways in the axis at ninety degrees. He was bored at a conference, and wrote in his notebook about putting two wheels at right angles to track movement on a plane. Years later, when he was searching for a device to select objects on a computer screen, he remembered those notes, and together with Bill English, he built the first mouse. We use the mouse not just because Doug Engelbart invented it, but because it turned out to be the pointing device that performed best for



pointing and clicking on a display, outperforming light pens, cursor keys, joysticks, trackballs, and everything else that was tried in early tests with users. The mouse won because it was the easiest to use.

We understand the reasons for the triumph of the mouse much more clearly from the story of developing the early designs told by Stu Card,[2] who joined Xerox Palo Alto Research Center (PARC) in 1974 and has spent much of his time there perfecting scientific methods to integrate with creative design. He has developed a process to predict the behavior of a proposed design, using task analysis, approximation, and calculation. His idea is to accelerate the movement through the design space by a partnership between designers and scientists, by providing a science that supports design. He tells the story of applying this science to the development of the mouse.

Why a Desktop?

■ Apple Mac OSX desktop with images of Apple mouse

Photo
Author's screen capture

It seems surprising to find a "desktop" on the spherical surface of a glowing glass display, housed in a bulky plastic box that in itself takes up half your desk. Or is it on the cramped flat screen of your laptop? Who came up with that idea? What were they thinking about, and why did they choose to design a desktop rather than a floor, or a playing field, or a meadow, or a river? Why does this desktop have windows in it? You usually think of windows being on the wall, not all over the surface of your desk. Why does it have a trashcan on it? It would seem more natural to put the trashcan on the floor.

In 1974 Tim Mott[3] was an outsider at Xerox PARC, working for a Xerox subsidiary on the design of a publishing system. He describes how the idea of a desktop came to him as part of an "office schematic" that would allow people to manipulate entire documents, grabbing them with a mouse and moving them around a representation of an office on the screen. They could drop them into a file cabinet or trashcan, or onto a printer. One of the objects in the office was a desktop, with a calendar and clock on it, plus in- and out-baskets for electronic mail.

There were lots of other people at Xerox PARC at that time thinking about desktops and other metaphors for use in the design of graphical user interfaces (GUIs), but Tim was working most closely with Larry Tesler,[4] and the two of them worked out processes for understanding users by talking to them, using guided fantasies, participatory design, and usability testing. Larry describes how he developed these processes and how icons arrived on the desktop. Larry insisted on simplicity and designed interactions that were easy to learn as well as easy to use. He went on to Apple and formed another partnership in the development of the desktop, working with Bill Atkinson[5] to create the designs for Lisa, including the pull-down menus, dialog boxes, and the one-button mouse. These ideas stayed in place as the user's conceptual model for the Macintosh and all of the GUIs that followed, stretching the desktop metaphor almost beyond the breaking point.

NLS, Alto, and Star

When Doug Engelbart invented the mouse, he arrived at the dominant design for input devices in a single leap from the light pen, and the development of the mouse since has been more in the nature of evolution than revolution.[6] Engelbart also invented the point-and-click text editor for the NLS system (oNLine System) that he developed at the Stanford Research Institute (SRI), and that system migrated with members of his design team to the fledgling Xerox PARC and became the foundation of the Alto, the first computer with a GUI. In the versions of NLS that were built at PARC for the Alto, the text-editing demonstrations were impressively fast, with a clattering of keystrokes that sounded businesslike and productive. Direct manipulation made it so easy to pick things up and move them that people would often find an instance of a word they wanted in the text and move it into place, instead of typing it. For programmers, this was a wonderful interaction, but the patience needed to acquire the skills proved a

fatal barrier for novice consumers when computers became accessible to ordinary people. It took the influence of Larry Tesler and Tim Mott to create a text editor and page layout design system that was really easy to use, and this was based on rigorous user testing and rapid iterative prototyping. They came close to the desktop metaphor that survives today.

One of the major innovations of the Alto was the bitmap display, making it easy to show graphics. You could make any dot be black or white, allowing you to put any image, whether a font or a picture, onto the whole screen. The memory to do that for a full-page display was exorbitantly expensive at the time, but it meant that you could translate the tradition of graphic and typographic design to the computer. Earlier computers had primitive and pixelated type fonts, but the bitmap display meant that what was on the display of the computer could look exactly like a book, except that it was only 72 pixels per inch instead of about three times as high a resolution for the printed page. A white background and dark text was used on the screen, so that the displayed image was like the printed result, and the screen size was chosen to be exactly the same as a page of text, enabling the concept of WYSIWYG (What You See Is What You Get).

The idea of the desktop was floating around Xerox PARC as part of the communal consciousness. David Canfield Smith had devised an office metaphor as part of his PhD thesis, "Pygmalion: A Creative Programming Environment," published in 1975. His icons looked like mathematical symbols and boxes that you could type into but defined the characteristics of the icons that have become commonplace. Here is his explanation:

> The entities with which one programs in Pygmalion are what I call "icons." An icon is a graphic entity that has meaning both as a visual image and as a machine object. Icons control the execution of computer programs, because they have code and data associated with them, as well as their images on the screen. This distinguishes icons from, say, lines and rectangles in a drawing program, which have no such semantics. Pygmalion is the origin of the concept of icons as it now appears in graphical user interfaces on personal computers. After completing my thesis, I joined Xerox's "Star" computer project. The

> first thing I did was recast the programmer-oriented icons of Pygmalion into office-oriented ones representing documents, folders, file cabinets, mailboxes, telephones, wastebaskets, etc. These icons have both data (e.g., document icons contain text) and behavior (e.g., when a document icon is dropped on a folder icon, the folder stores the document in the file system). This idea has subsequently been adopted by the entire personal computer and workstation industry.[7]

When Dave Smith had finished his PhD, he was hired to join PARC to work on the Star system. Alan Kay had always talked about the desktop metaphor and he came up with the idea of overlapping windows, from a metaphor of papers on a desktop. "You can have a lot more papers on a desk if they overlap each other, and you can see the corner of one, pull it out and put it on top."

Detailed accounts have been written[8] about how long it took to convince a copier company to commercialize the Alto, but Xerox did make a major effort to develop and market an "Office of the Future" machine. They did a thorough analysis of the business potential to find what the requirements of that machine should be, concluding that people would pay a lot for the Alto technology. Star was conceived in response to this; based on the combination of Smalltalk and Alto, it became a design that fit all of the requirements. Star was a very futuristic machine; when they were asked in the market research surveys, people responded that they would not give up that advanced interactive performance for a much inferior but less expensive machine. After it was launched, the IBM PC came out, and the people who said in the research that they would pay a lot for Star proved to be only willing to pay much less for an inferior interface.

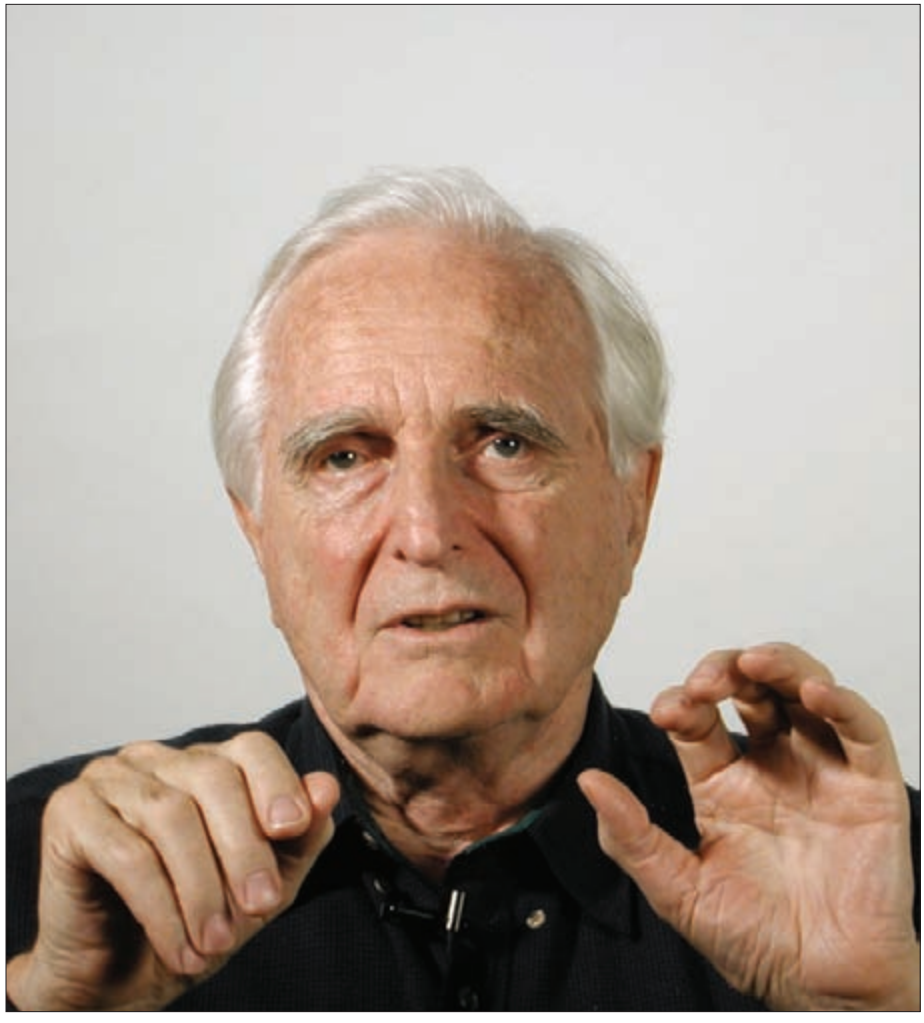
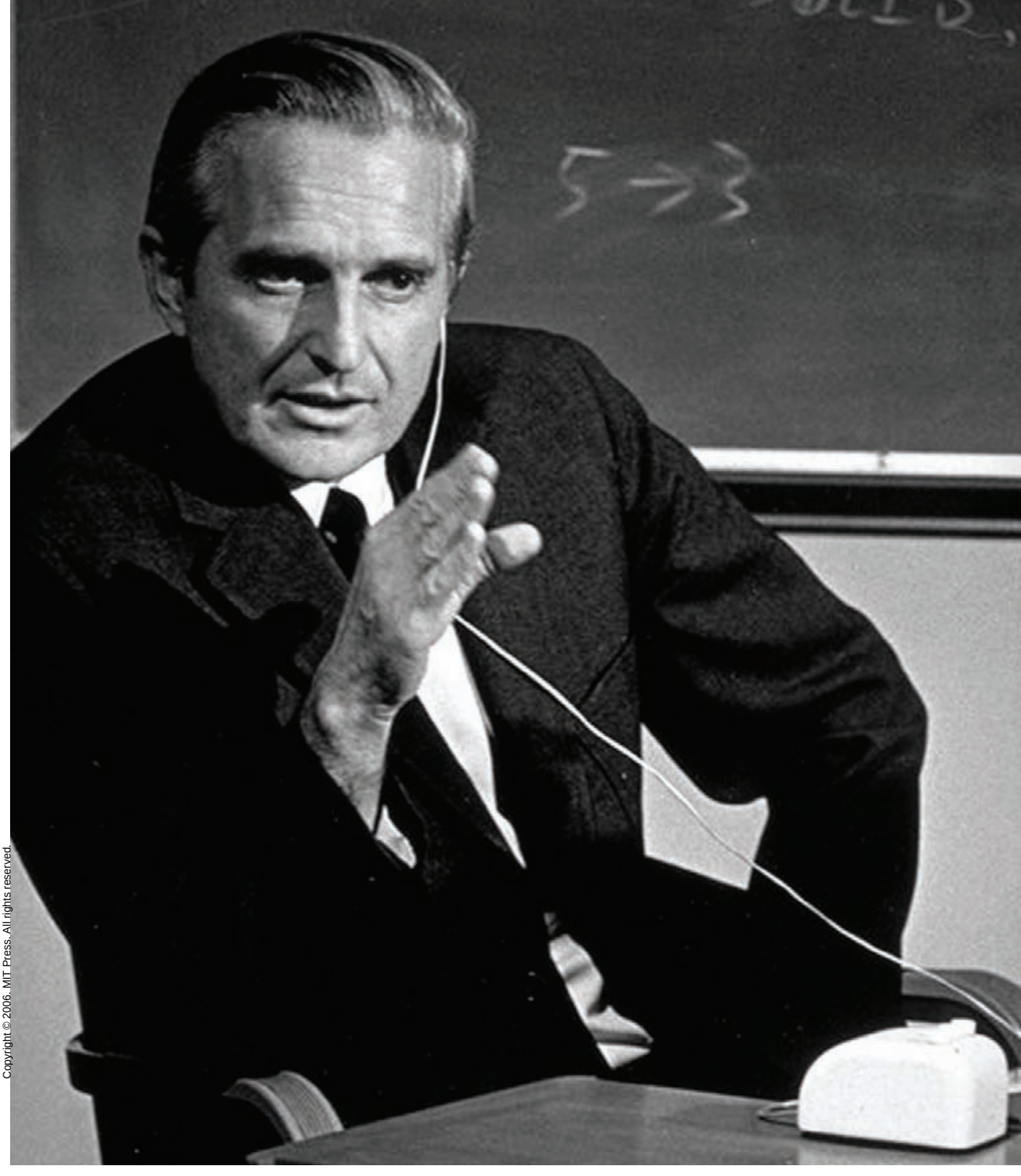


Photo Author

Dr. Douglas C. Engelbart

Doug Engelbart is best known as the inventor of the mouse. At the time of writing, in 2004, at the age of seventy-eight, he is still going to work at his "Bootstrap Alliance," trying to persuade us of the value of his ideas about augmenting the human intellect, and evangelizing the virtues of his "Augment System." His office is located in the headquarters building of Logitech, the world's largest manufacturer of mice. Taking credit as the inventor of such an ubiquitous product is not enough for Doug; in fact, he is charmingly modest about that achievement, preferring to discuss other ideas that have not met with such spectacular success. He has always wanted to create designs that enhance human performance and is not interested in ease of use for the novice. He grew up in Oregon, and his electrical engineering studies were interrupted by service in World War II. During his stint in the Philippines as a naval radar technician, he came across an article by Vannevar Bush in the *Atlantic Monthly*[9] and was inspired to think of a career dedicated to connecting people to knowledge. This idealistic ambition led him to the Ames Laboratory (later NASA Ames Research Center) on the edge of the San Francisco Bay, where he worked on wind tunnel research, and then to the Stanford Research Institute, where he invented the mouse and built up the Augmentation Research Center (ARC) with funding from ARPA. In the early seventies he took several members of his team to Xerox PARC, where he helped put the mouse and the desktop together.



Doug Engelbart

Inventing the Mouse

In 1995 the International World Wide Web Conference committee presented the first SoftQuad Web Award to Dr. Douglas C. Engelbart, to commemorate a lifetime of imagination and achievement and to acknowledge his formative influence on the development of graphical computing, in particular for his invention of the mouse. This is the contribution for which Doug is universally acclaimed. In spite of this, he himself is inclined to give credit to the trackball:

■ Doug Engelbart conducting a workshop circa 1967–68

Photo
Bootstrap Institute

> When I was a senior in electrical engineering, some of the experiments we had to do in the laboratory would end up resulting in funny shaped curves that curled back on themselves, and it was the area under the curve that we were experimenting with. They had an elbow shaped device there, a platform sort of thing that would set down on the table. You would run the pointer around that area, and there was a small wheel next to the pointer, resting on the tabletop, and the other side of the joint there was another one. I couldn't figure out how that would produce the area under the curve.
>
> "I'm not sure myself about the mathematics," said the professor, "but it is a fact that the little wheels will only roll in the axis of the rolling direction, and will slide sideways."

I began to understand that the wheel would roll only as far as you went in the one direction, irrespective of how many sideways movements you made.

I thought of that again one time during a conference on computer graphics, when I was feeling rebellious and bored, so I wrote in my notes about putting two wheels at right angles to each other, so that one would always be measuring how far you went north and south, and the other east and west. It would be easy to convert that into potentiometers and things so that the computer could pick up that signal. It was a very simple idea. At the time I was unaware that that same thing was sitting underneath the tracking ball, and that was how a tracking ball worked. Later on the manufacturers put the wheels against the table, which is exactly like an upside down tracking ball. There should be credit given to the tracking ball, except for my ignorance about it at the time.

It says a great deal about Engelbart's extraordinary modesty that he makes so light of his achievement. It also says a lot about his methodical persistence that he used his moments of boredom at that conference to fill a notebook with ideas and that he remembered what was in that notebook when he was looking for the best input device solutions many years later:

When you were interacting considerably with the screen, you needed some sort of device to select objects on the screen, to tell the computer that you wanted to do something with them. We got some funding in the early sixties, I think it was from NASA, and set up an experimental environment with several different kinds of devices; a tracking ball, a light pen, and things of that sort that were available at the time.

As we were setting up the experiments, I happened to remember some notes that I had made in a pocket notebook some years before, and sketched that out to Bill English, who was the engineer setting up the experiments, and he put one together, with the help of a few draftsmen and machinists. That one was put in the experiments, and happened to be winning all the tests. That became the pointing device for our user interface. Somebody, I can't remember who, attached the name "mouse" to it. You can picture why, because it was an object about this big, and had one button to use for selection, and had a wire running out the back.

■ Graficon experimental pointing device

"It looks like a one-eared mouse!" someone said. Soon all of us just started calling it a mouse.

Thinking of the possible relevance of those orthogonal wheels was the first step; working with Bill English to design an object to contain them was the second. The recognition that this idea might be important for interaction came from the tests that compared the mouse with other possible input devices; it was the people who used it in the tests who proved the point. The designers came up with as many alternatives as they could that seemed plausible, built prototypes and created tasks in the relevant context, and then ran the tests. Here's what Doug says about the testing process:

> We listened to everybody who had strong ideas, and it seem to us worth just testing everything that was available. The light pen had been used by radar operators for years and years, and that seemed to most people would be the most natural way to do it. I couldn't see that, but why argue with them; why not just test and measure? The time it takes to grope for it and lift it up to the screen seemed excessively large, so it didn't do well in the tests.
>
> For the test we had naive users coming in, and we explained everything that would happen so that they weren't surprised. We asked them to put their hands on the keyboard, and all of sudden an array of three-by-three objects would appear at an arbitrary place on the screen, sometimes small objects and sometimes large, and they had to hit a space bar, access the pointing device and go click on it. The computer measured time, overshoot, and any other characteristics we thought were valuable. The assessment just showed the mouse coming out ahead. It was many years later that I heard from Stuart Card, a friend at Xerox PARC, what the human factors explanation was.

There is an objectivity in this process of letting the user decide, the value of which is a recurring theme in this story of designing the desktop and the mouse. Come up with an idea, build a prototype, and try it on the intended users. That has proved, time and time again, to be the best way to create innovative solutions.

First mouse in hand, 1963–64 ■
First mouse ■
First production mouse ■

The Demo that Changed the World

There is a gentle modesty, even diffidence, in the way Doug Engelbart talks, but he holds your attention much more firmly than you would expect from his manner of speech. His passion for philosophy and ideas shines through, with an underlying intensity, almost fanaticism, that is charismatic. He remembers his early motivations:

> My initial framework for thinking about these questions got established in 1951. I had realized that I didn't have any great goals for my career. I was an electrical engineer with an interesting job, recently engaged to be married, but had no picture of the future, and I was embarrassed about this.
>
> "What would be an appropriate career goal for me?" I asked.
>
> "Why don't I design a career that can maximize its benefit to mankind?" I ended up saying.
>
> I was an idealistic country boy. Eventually I realized that the world is getting more complex at an ever more rapid rate, that complex problems have to be dealt with collectively, and that our collective ability for dealing with them is not improving nearly as fast as the complexity is increasing. The best thing I could think of doing was to try and help boost mankind's capability for dealing with complex problems.

By this time he had been working for a couple of years at the Ames Laboratory, in what is now the heart of Silicon Valley but was then still a pleasant agricultural countryside full of orchards. His job researching aerodynamics and wind tunnel testing was interesting and enjoyable, he was engaged to the girl of his dreams, and life might have been good enough; then that idealistic itch to change the world took over, and he started his lifelong search to develop electronic systems that would augment the human intellect. He remembered the "Memex" that Vannevar Bush had described as an "enlarged intimate supplement to a person's memory" that can be consulted with "exceeding speed and flexibility." He felt a kinship for the vision and optimism that Bush communicated and set out to find his own way of realizing an equivalent ambition.

> When I was half way through college, I was drafted for World War II, and had the good fortune to get accepted in a training program that the navy was running for electronic technicians, because the advent of radar and sonar had changed the aspects of navy problems immensely. They had a year-long program which taught me a lot of practical things about electronics and exposed me to the fact that the electronics of radar could put interesting things on the screen, so I just knew that if a computer could punch cards or send information to a printer, then electronics could put anything you want on the screen. If a radar set could respond to operators pushing buttons or cranking cranks, certainly the computer could! There was no question in my mind that the engineering for that would be feasible, so you could interact with a computer and see things on a screen. That intuitive certainty made me change my career path totally to go after it, but I had an extremely difficult time conveying that conviction to anybody else successfully for sixteen years or more.

His first step in that sixteen-year path of dogged determination was to leave his job and go to the graduate school at the University of California at Berkeley, where one of the earliest computers was being constructed. His fixed idea that people should be able to interact with computers directly did not fit with the prevailing view, so he started to get a reputation as an eccentric. Once he had his PhD, he started to look around for a place that would be more accepting of his vision than the UC Berkeley community. He talked to Bill Hewlett and David Packard, but although they were enthusiastic about his ideas, they were determined to focus on laboratory instruments rather than computers.

He finally landed a job at SRI, whose leaders were interested in researching possible uses for computers in both military and civilian applications. He started there at the end of 1957, soon after Sputnik had been launched, and the space race was getting under way. After learning the ropes at SRI for a year and a half, he started to lobby for the opportunity to start his own lab to experiment with new ways of creating and sharing knowledge by combining man and machine. His wish was granted when the U.S. Air Force Office of Scientific Research provided a small grant, and he settled down to the task of articulating his views.

"I wrote a paper that was published in 1962 called 'Augmenting the Human Intellect: A Conceptual Framework'[10] that steered my life from that point forward." In his paper he defined four areas in which human capabilities could be augmented:

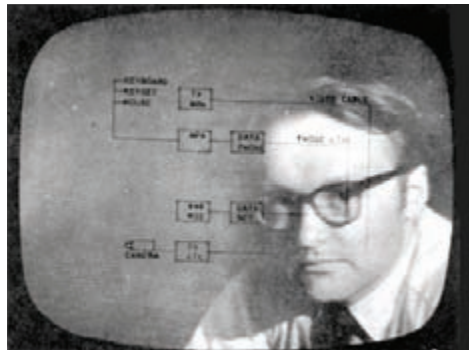
> 1. **Artifacts**—physical objects designed to provide for human comfort, the manipulation of things or materials, and the manipulation of symbols.
>
> 2. **Language**—the way in which the individual classifies the picture of his world into the concepts that his mind uses to model that world, and the symbols that he attaches to those concepts and uses in consciously manipulating the concepts ("thinking").
>
> 3. **Methodology**—the methods, procedures, and strategies with which an individual organizes his goal-centered (problem-solving) activity.
>
> 4. **Training**—the conditioning needed by the individual to bring his skills in using augmentation means 1, 2, and 3 to the point where they are operationally effective.
>
> The system we wish to improve can thus be visualized as comprising a trained human being, together with his artifacts, language, and methodology. The explicit new system we contemplate will involve as artifacts computers and computer-controlled information storage, information handling, and information display devices. The aspects of the conceptual framework that are discussed here are primarily those relating to the individual's ability to make significant use of such equipment in an integrated system.

In this short quote one can see the seeds of triumph and tragedy. The triumph is Doug's powerful vision of a complete system, where people and computers are engaged in a symbiotic relationship for human benefit, working cohesively as an integrated system. From this came the mouse and the other elements of interactive computing that he pioneered. The tragedy is that training is a necessary component of the system. He developed concepts for experts, and the pursuit of the highest capability drove the design criteria; it was therefore inevitable that training would be needed to reach the level of proficiency that

would let people benefit from this capability. That proved a barrier to acceptance by ordinary people, and as computers became less expensive and more accessible, the barrier got in the way more and more.

But we are getting ahead of ourselves in the story. Let's go back to 1964 when, to the surprise of the SRI management, the Defense Advanced Research Projects Agency (DARPA) offered to fund the Augmentation Research Center (ARC) to the tune of half a million dollars a year, as well as providing a new time-sharing computer system, worth another million. Engelbart's energetic lobbying for funding and his flow of papers describing the high-level potential of automation had not been ignored by everyone, so now he had the resources he needed to move from theory to practice. He put together a stellar team of engineers for both hardware and software and set about developing NLS. Bill English was a partner for Doug in much of the work they did, leading the hardware development as the team grew to seventeen people. He joined ARC in 1964 and was the perfect complementary talent, having the technical ability to implement many of the ideas that were expressed by his boss as high-level abstractions. After four years of development, Doug took a chance to show the computer science community what he had been doing:

> For the Fall Joint Computer Conference in 1968, I stuck my neck out and proposed giving a real-time demonstration, if they would give me a whole hour-and-a-half conference session. We had a timesharing computer supporting our laboratory, and small five-inch diagonal screens that were high resolution, but worked by moving the beam around (vector graphics). We put a TV camera in front of the screen and used a TV display for a larger size image. We rented microwave links (from the Menlo Park civic auditorium) up to San Francisco, and borrowed an enormous video projector, three feet by two feet by six feet, to project onto a twenty-foot screen for the audience to see, using a very novel way of converting the video sweep into modulated light.
>
> Bill English, the genius engineer that I worked with (on the mouse also) managed to make all this work. He built a backstage mixing booth, where he could select from four video feeds, one from



Bill English ■

each of the linked displays, one looking at me, and one overhead showing my hands working. He could select from the feeds so you could see a composite image in real time. He had experience doing the stage work for amateur plays, so he was the director. I had written a script for different people to come onto the stage, so he put a speaker in my ear to let me hear his cues: sometimes it was so distracting that I would fumble words.

We were able to show high-resolution links, graphics for schematic diagrams of what was going on, the faces of members of our team in the Menlo Park Laboratory, as well as the screens that they were looking at. We had cursors controlled by two people simultaneously interacting on the screen; one guy started buzzing at my cursor as if in a fight. The audience all stood up and applauded at the end of the demo.

This was the demo that changed the world. The computer science community moved from skepticism to standing ovation in an hour and a half, and the ideas of direct manipulation of a graphical user interface became lodged in the communal consciousness. This was not punched cards and Teletypes. It was something entirely different. Doug sat alone at a console in the middle of the stage, with the twenty-foot screen behind him showing the view from the video feeds. He was wearing a short-sleeved white shirt and a thin tie, with a microphone dangling on his chest, and a headset like an aircraft controller's. The overhead camera showed his right hand using a mouse, seen for the first time by most of the audience, to point and select with; a standard typewriter keyboard in the center and a five-key command pad under his left hand. In his calm but mesmerizing voice, he described and demonstrated an amazing array of functions on the NLS system. Words were manipulated with full-screen text editing, including automatic word wrap, corrections and insertions, formatting, and printing. Documents were planned and formatted using headings and subheadings. Links were demonstrated between one document and another, and collaboration between remote participants was demonstrated in real time:

One of the basic design principles that we started with was that you want to be able to talk about any other knowledge object out there.

You want your links to point in high resolution, for example to a given word in another document, so you want a link addressing string that will let you get there. You also should be able to have optional alternative views.

"I just want to see that one paragraph," I might say.

"Okay! When I get there, I'd like to have certain terms highlighted."

"Okay! I'd also like to know who else in my group has links to it, and what they say about it."

I wrote in 1962 that we are all used to the idea that we can diagram sentences, based on the syntactical rules for a properly structured sentence. Now we might want to see a similar set of rules for the structure of an argument, so the computer has a graphic diagram of the argument with nodes connected to other arguments, expressions and statements, and meaningful links connecting them.

The demo was truly amazing, proving that interactive computing could be used for real-time manipulations of information in ways that very few people had imagined before. The assumption that high levels of training would always be acceptable did not get in the way until ordinary people tried to become users of NLS.

The demo also positioned Doug and his band at ARC to receive continuing funding for their research until 1975. His team grew to thirty-five people at one point. In 1969 they were connected to ARPAnet as one of the original nodes of the military research connected network, which eventually developed into the Internet. NLS grew in sophistication and content as time went on but remained essentially the same in concept. In 1971 a group of the best people at ARC, including Bill English, were tempted away from SRI by the opportunities at the new Xerox PARC, where so many exciting things seemed to be about to happen. This was the start of a slide for Doug Engelbart, during which his long-held dreams seemed to have less and less influence. You can feel the frustration behind his words as he describes the determined pursuit of his ideals and bemoans the success of the desktop:

I've been pursuing for fifty years something that required higher and higher levels of capability. "How do you achieve capability," I was

Demo at Joint Computer Conference, 1968 ■

asking, "and when you achieve capability levels as high as you can get, then how do you reduce the learning costs in a reasonable way, but not try to set what I think of as an artificial level of learnability ease, and have to keep your capability enhancements within that level?"

In the business world, I understand, that is awkward to try to do, because you are competing with other people for sales, and people will try computer interfaces that will strike customers as easy to use early on when they purchase them. I don't object to having a difference, but I feel that the world should recognize that there are really high levels of capability there to pursue that will be very important for society to have reached. That's been my pursuit.

Many years ago it became clear to me that what you need to do is develop a basic software structure that will have file designs, capabilities, and properties that the very expert person could use. Then it is easy enough to support the beginner, or pedestrian user, by plugging a very simple user interface with simple operations on the front, but they can both work on the same materials.

Yes, you can point with a GUI, I admit, but our system had an indefinite number of verbs and nouns that you could employ.

There's no way that pointing and clicking at menus can compete with that. You wouldn't want to give someone directions by that limited means.

It is easy to understand the idea of going for the best, of catering to the expert user, and then providing a path to get there from a simple user interface designed for the beginner. In practice, however, this has proved to be the wrong way round, as it's not easy to get something right for the beginner when your design is already controlled by something that is difficult to learn. Look, for example, at the use of the five-key keypad for typing text. Like the stenographer's keyboard used for recording court proceedings, it enables impressive typing speeds when you have been trained long enough to become expert.

Quite a few users adopted the chording key set that I built for myself. You could type any of the characters in the alphabet with one hand, and give commands with the three-button mouse in the other hand at the same time as pointing. This gave a much richer vocabulary and a much more compact way to evoke it than the GUI.

This is how the interactions were designed. On the mouse, one button was to click, another was called *command accept*, and the third was called *command delete*. If you wanted to delete a word, you would hit the middle button on the keypad, which was the letter *d*. It was *d* because it is the fourth letter in the alphabet, and this was a binary coding, 1, 2, 4, 8, 16. If it was the letter *f*, it was the sixth letter, so you'd hit the 2 and the 4 keys at the same time. Then you pointed at and clicked the beginning of the thing you wanted deleted, then you pointed at and clicked the end of the thing you wanted deleted, and if you hadn't made any mistakes, you would hit the *command accept* key on the mouse. It was *d*, point/click, point/click, *command accept*. If you made a mistake at any point, you would hit the *command delete* key to back you up one step.

That process was complicated to learn, and it took a long time for most people to memorize the binary-based, five-finger alphabet. Alternatively, they could invoke commands with keypad, or even the keyboard, the mouse for pointing, and a conventional keyboard in between for typing text. This would have been a very good solution for people with four hands, but is not as fast as the chorded keyboard and three-button mouse, as it takes longer to move your hands to and from the keyboard.

Doug Engelbart strives consistently toward a goal of the best possible performance, and his intuitions and insights have set the scene for the dominance of the mouse and the desktop. His influence has been limited by his decision to design for people as determined and proficient as he is himself, rather than for those who require an easy-to-use system.

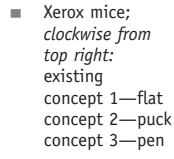
Photo Author

Stu Card

At first meeting, Stu Card seems to be a serious person. He looks at you intensely from behind his glasses and speaks in bursts, often veering off on a new tangent of connected thought. You have to concentrate hard to keep pace with him, but when you do, the reward is immediate, as he has thought everything through and arrived at a beautifully balanced view of the whole picture. Occasionally his face breaks into an impish grin, and you see that there is a rich sense of humor under the seriousness. He joined Xerox PARC in 1974, with probably the first-ever degree in human-computer interaction. As a freshman in high school, Stu built his own telescope, grinding the mirror himself. His ambition to be an astronomer led him to study physics, but he was always very interested in computers. In the eighth grade he read navy circuit manuals about how to build flip-flops out of vacuum tubes. His first evening course in computing was to aim the college telescope in the direction defined by a computer program that he wrote. At graduate school at Carnegie Mellon University he had designed his own program, studying computer science, artificial intelligence, and cognitive psychology. After graduation he was offered three jobs; PARC was the most interesting, had the highest salary, and was in California, so it was an easy decision. Doug Engelbart and Bill English had brought the mouse to PARC from SRI, and Stu was assigned to help with the experiments that allowed them to understand the underlying science of the performance of input devices.

Stu Card

A Supporting Science



Xerox mice; *clockwise from top right:* existing concept 1—flat concept 2—puck concept 3—pen

When Stu joined PARC in 1974, he set out to invent a supporting science for the design of human-computer interactions. This made his position somewhat different from the other researchers. There had been several attempts to develop similar sciences, for example human factors, but that focused too much on the evaluative side, waiting until the structure of the design was already complete and then measuring the result. Stu was more interested in contributing to the design process at the beginning, when the significant choices were still in flux and the science would be able to influence the outcome before much work was complete:

> Newell, who was a consultant at PARC at the time, wanted to try to do a kind of applied psychology. The idea was that computer science is a very asymmetrical discipline. If you take things like time-shared operating systems or programming languages, there is obviously a computer part to them, like how you parse them and do the compiling. There is also obviously a human part; what kind of languages can people program most easily, and what kind of errors do

they make, and how can you make them more efficient? All of the work that had been done in computer science was on the computer side and none on the applied psychology side. Information processing psychology showed real promise as a theory, but there were problems with psychology; it tends to be faddish and impractical and hard to build on.

The idea was that any science worth its salt should have practical applications, allowing you to test the theory in a more pragmatic context than a journal article. It would really have to work, as you were not immune to the hostile reactions of the people that you were trying to do this for. It would be difficult to do this in a university, because universities, especially psychology departments, would not tolerate applied work. In order to do basic research, you had to do it in a place like PARC.

"Design is where all of the action is!" was one of our slogans.

If this was going to work at all, you had to have something that could be used as you were designing. This did not mean that you could do all of design from science. You could have the equivalent role to that of structural engineering in relation to architecture. You could have a technical discipline that would support the design activity. In fact, we had a particularly concrete notion of what kind of supporting science this would be; task analysis, approximation, and calculation. The idea was that you would be able to look at a situation, make zero parameter computations about it, and then say things about what would happen in that situation without running full experiments to see, rather with just occasional experiments to spot check. The idea was that this would give you lots of insight that it would otherwise be hard to have gotten.

"Don't bug us for ten years," we said, "and at the end we'll deliver this to you."

The wonderful thing about PARC was that they said, "Sure." We had to make arguments up front, but once they were made, we got ten years to do this. I've always appreciated the freedom that I got to do this at PARC.

To think that design is where all the action is was very forward-looking at that time, when innovation was usually thought to come from genius or pure scientific research. The core skills of design are synthesis, understanding people, and iterative prototyping. Even now, the idea that these skills are central to innovation is not very widely accepted, but you will see evidence

of their significance again and again throughout this book. The unique contribution that Stu Card made was to create a supporting science, connecting the theoretical underpinnings of research to the pragmatic synthesis of design.

Bill English was Stu's first boss at PARC. Bill and Doug Englebart had coinvented the mouse when they were at SRI, with Doug providing the idea and Bill the engineering development and prototyping. A large part of English's group had come over to PARC from SRI. They wanted to do more experiments on the mouse to determine whether it or some other device was really the better one. There were devices like rate-controlled isometric joysticks, almost identical to the one in the IBM ThinkPad keyboard today; there were various kinds of trackballs; and there were many versions of buttons and keys. Stu remembers the experiments:

> English was pretty busy, so I agreed to help him do the experiments. He set up the usual kind of A-versus-B experiments between devices, but the problem with them was that he did not really know why one was better than the other, as the results varied when you changed the context. Since we were trying to do the science of this stuff, I modeled each one of the devices so that I had the empirical differences between them, and I was trying to figure out how numerically to account for why those differences occurred. The most interesting model was the model of the mouse.
>
> Fitts's[11] law says that the time to point goes up as the log of the ratio between the distance and the size of the target. What's interesting about this law is that the slope of that curve is about ten bits per second, which is about what had been measured for just moving the hand alone without a device. What's interesting about that is that the limitation of the device is not then in the device itself, but in the eye-hand coordination system of the human. What that meant was that the device was nearly optimal. The hand was showing through the machine instead of operating the machine at a loss, and so if you were to introduce this onto the market, nobody would be likely to come up with another device to beat you. We could know this theoretically once we had this one empirical fact.

Stu went down to El Segundo to meet the Xerox engineers who were trying to invent a pointing device for the office system

- Xerox Alto
- Xerox Star
- Xerox Star mouse

that was planned. They resisted the idea of something outside the normal cabinet full of standard electronic racks that needed a work surface and a connection cable and would have to be packaged separately. Stu presented the results of his tests, with a lot of interruptions.

"Why didn't you run it this way?"

"Who were your subjects?"

"Why didn't you have eighty-year-old grandmothers try it?"

There was some flack and hostility, but when Stu gave his theoretical explanation of his supporting science, the room fell silent and he won the day. Xerox put a mouse on the market, and what Stu had predicted was true. It is still the most successful pointing device to this day.

In introducing the Star system, which was the next step after the Alto toward commercialization, there was a certain circuit in the machine, and Stu was asked whether the circuit would be fast enough to track the mouse. If they had to make it faster, they would have to rip out the existing circuit and put in a more expensive one. Stu discussed this over lunch one day with a colleague, and because he had a theory of the mouse he was able to whip out a napkin and make a calculation, which indicated that the circuit was going to be too slow. He went into the laboratory and took a little video of a mocked up task to check the theoretical curve, and that confirmed the problem, so they had to change to the more expensive circuit. It was possible to arrive at this definitive result four hours from the initial formulation of the problem because there was a theory. There was no need to run lots of user experiments and counterbalance them and do the analysis. In a design context you want to very rapidly say, "I believe it should be thus because of such a thing," and then do a little checking to be sure that you didn't leave something out.

Stu demonstrated that you could use the same theory to understand how to develop a better pointing device:

> Just as we showed the advantages of the mouse, you could use the same theory to show that you could beat the mouse. The mouse was optimum, but it was based on movements of the wrist. If you looked at Langoff's thesis, he had measured Fitts's law slopes for different

sets of muscles. In particular, when you put the fingers together you can maybe double the bandwidth.

"That's where you want to put your transducers!" I said.

If you see these maps of the motor cortex and how much of it is devoted to different sets of muscles, you want to put you transducer in an area that is covered by a large volume of motor cortex. You could probably do one with the tongue too, because there's a lot of space devoted to that. That's another example of how having some notion of the theory of this thing gives you some structuring of the design space.

In the mid eighties ID TWO[12] was engaged by Xerox to design the enclosures for a new workstation and wanted to push the design of the mouse farther. Stu Card was able to help the designers think about new concepts for pointing devices. He demonstrated one idea with a pencil stuck onto an eraser, to make it easier to grasp when you are only seeing it out of the corner of you eye. He believed that it had become feasible to build a mouse like this because of the purely optical mouse that had been developed at PARC by this time. The whole mouse was on a chip, so most of the case was empty, and that meant that you could shape the casework to be optimal to the shape of the hand, instead of needing to make it big enough to cover the mechanism.

As soon as Stu explained how to structure the design space from his theory, the designers were able to quickly create alternative designs for pointing devices. They produced three models, based on progressively more radical assumptions. One was a conventional mouse that was flatter and more elegant, the next was a puck shape, like a top hat of about an inch diameter, with a skirt at the base for the fingers to rest on. The most radical one was the penlike device with a weighted base, causing it to stand upright on the desk, giving form to Stu's idea of the pencil stuck into the eraser. At the end of the project Stu concluded:

> This was my ideal model of how the supporting science could work. It required good designers to actually do the design, but what we could do was help structure the design space so that the movement through that design space was much more rapid. The science didn't design the mouse, but it provided the constraints to do it.

Xerox puck and pen mouse concept designs ■

Tim Mott

Tim Mott created the concept of "guided fantasies" to learn about user needs, and was one of the very first people to apply rigorous user testing to the design of user interfaces. He studied computer science at Manchester University in England in the sixties and found a job with a publishing company called Ginn, near Boston, that was owned by Xerox. This led him to work with the researchers at Xerox PARC, and he collaborated with Larry Tesler to design a publishing system that included a new desktop metaphor; together they invented a user-centered design process. In the late seventies he became more interested in designing processes for management and business and honed his management skills working for Versatec, another Xerox subsidiary that manufactured electrostatic printers and plotters. In 1982 he cofounded Electronic Arts (EA) and set about building a set of processes to enable the creation and production of really rich interactive entertainment experiences—as soon as the supporting hardware was available. From the very beginning, they built the company with people who were just crazy about games. Once EA was successful, Tim went on to run a small company called Macromind, whose founders had invented a user interface design tool called Director, leading the company to expand into multimedia and become Macromedia. He was a founding investor in Audible, setting the precedent for the MP3 players that came later, and moving from "Books on tape" to the spoken word Web site that supports public radio. He is a pilot, flying jets as well as the single-engine back-country plane that he loves to fly over wild mountain scenery.

Tim Mott

Tim put the editors in front of a display with a keyboard and a mouse. Nothing was on the display and no programs were running, but he asked them if they could walk him through the process, imagining what it would be like to use that hardware to edit. This was 1974, before word processors, so they were using typewriters, pencils, and erasers.

■ Tim Mott in 1974

Guided Fantasy

GINN WAS ONE of several companies in the publishing industry that Xerox acquired around 1970. They were based in Lexington, Massachusetts, in one of the first buildings equipped with office systems furniture from Herman Miller. When the Ginn management team found out how their contribution to Xerox corporate finance was being spent, they put in a request:

> We're taxed for your research center; every division of Xerox has to pay money to support Xerox PARC research, and we want something back! What are you going to do for us?

This challenge eventually reached Bill English, who assigned the task of developing a publishing system for them to Larry Tesler. Larry started working with Ginn and writing specifications for their system and suggested that they hire somebody; they could send the new person to PARC for a year or so to work on the design of the new product. As it moved from design to implementation, he could write some of the code and then return to Ginn to provide support. They found Tim Mott at Oberlin College, where he was working on the help desk for a mainframe. Larry remembers interviewing Tim:

> When we talked to him, we realized that this was the perfect guy; he writes good code, he's really fast, he understands about usability because he helps customers all the time, he loves doing support and he loves programming, and he thinks it's an interesting problem. We hired Tim Mott, and we got ten times more than we bargained for.

Tim spent a little time at Ginn, observed how they worked, started thinking about how to make it easier for the publishing team to get their job done, and then flew out to PARC. He remembers his first encounter with POLOS (PARC On Line Office System):

> When I got out there, what I found was that, in my judgment at any rate, the system was completely unusable by anyone other than the people who built it. Their background was in building editing systems and design systems for themselves and for other computer professionals, and at least in my judgment it just wasn't going to be usable by editors and graphic designers who worked in a publishing company, so a month after getting to Palo Alto, I wrote a letter of resignation to the executive editor back in Boston, saying, "This is not a project that you should be doing." That letter found its way to Bob Taylor, who was running the computer science lab at the time. I spent some time with Bob, and he said, "Why don't you figure out what it is that we should be doing then?"

That was a challenge that Tim couldn't resist, so he teamed up with Larry Tesler, and together they set about discovering what the people who worked at Ginn really needed. Tim went back to Lexington and put the editors in front of a display with a keyboard and a mouse. Nothing was on the display and no programs were running, but he asked them if they could walk him through the process, imagining what it would be like to use that hardware to edit. This was 1974, before word processors, so they were using typewriters, pencils, and erasers. He used the "guided fantasy" technique that Larry had told him about, explaining that the mouse could be used to position a pointer on the screen and that the text would be on the screen. The editors described the process that they used at that time with paper and pencil. Together they imagined typing in the text and creating a manuscript, and

then editing that manuscript using the mouse and keyboard in the same way that they would use a pencil.

The Alto and the POLOS in 1974 both had mice on them based on the Engelbart mouse developed at SRI, and the Alto display was already a bitmap display, but the design of editing programs was still based on the prior generation of character displays; no one had really thought about how to use the characteristics of the bitmap display to create a more flexible editing environment.

None of the text editors that had been designed for computers up to that point had a space between characters. They were based on character matrix displays that didn't allow for an editing mark to be placed between the character cells. If you wanted to mark an insertion point in the text, you either selected the character before where you wanted to add the new text, and said *append*, or you selected the character afterwards and said *insert*. Tim explains the difficulty that this caused:

> That was one of the first things that I got from working with these editors. One of the concepts that they worked with was the space between characters, or the space between words.
>
> "I want to use the mouse and put this insertion point or this caret between these two characters, and then I just want to type in the new text," they said.
>
> When it came to deleting text, they talked about wanting to strike through it, just like they would with a pencil. They wanted to use the mouse to draw through the text. Up until that point, the way that a span of text had been selected was to mark the beginning point, and mark the end point and say "select." No one had actually used a mouse to draw through text.
>
> These techniques that we see in all the word processing programs today came directly from working with people who spent their entire lives editing text and asking them, "How do you want to do that?" Once Larry and I hit on this idea of having people talk about how they would want to do the work, the design itself became pretty simple. There was a new design methodology that came out of it. We talked about the design process as one that began with building a conceptual model that the user had today.

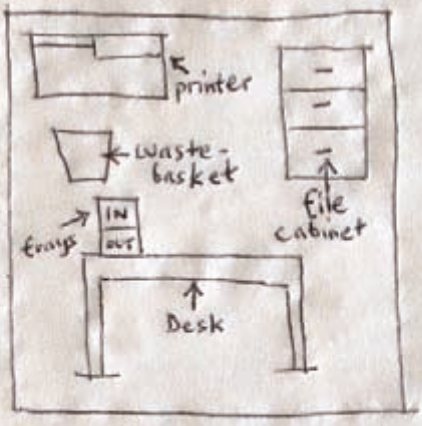
The Alto didn't have a user interface; rather, it came with a toolbox. You could build any kind of program you wanted, based on this very flexible architecture, and people started building applications on it. The Smalltalk system incorporated menus and editing techniques. There was a graphics program that William Newman built called Markup and a paint program that Bob Flegal created—applications started springing up.

Tim and Larry took the code base from Bravo,[13] a text editor that already existed at PARC, and put a completely different user interface on top of it. They called their design "Gypsy"; the implementation program was straightforward, but they worked hard. They shared one Alto between them, working fourteen-hour shifts so that they would overlap by an hour at both ends and could tell each other what they had done. In this way they could work on the same code around the clock and could protect their access to the computer, as there were only four Altos at PARC when they started.

Using "drag-through" selection for text was not the only innovation that came out of the work on Gypsy. The origin of the cut-and-paste metaphor is described by Larry in his interview, as is Tim's idea for double-clicking the mouse. There was also the first dialog box; this was a little bar with a place for typing commands such as *Find*. It was more like a noniconic toolbar of today, as in Apple's Safari browser.

Gypsy had a file directory system with versions and drafts, not just a list of files. You could have versions of the document and drafts of the version; it remembered them all and organized them all for you. It had bold, italic, and underline—used very much like they are today. You could select something by dragging through it or clicking the two ends, and do the equivalent of *Control b* for bold. They changed the name of the control key to *look* with a paper label, so you said *look b* for bold.

Larry had come up with the "guided fantasy" technique in 1973, and he and Tim developed other user-centered methods with frequent usability testing in the fall of 1974 and the spring of 1975; their work really caught the imagination of the people in the PARC community. In the spring of 1975 Tim took the text-



Office Schematic

PRINT, FILE, DELETE, MAIL

↓

all are inter-doc actions

—11—

INTRA-DOC use cut & Paste physical metaphor

Whats analog for INTER-DOC ??

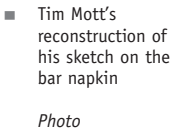
↓

Grab & Move !!!

editing program back to Lexington to try it out with the people at Ginn. For the first test he picked the most senior editor, figuring that, since she was the most entrenched both in her work habits and the company, if he could win her over, it would be smooth sailing from then on. After the first day she said, "You know, I think the quality of my work will be better going forward, because it's just so much easier to edit with this than it is with a typewriter and a pencil!" The approach had proved itself!

The Desktop (Office) Metaphor

In parallel with completing the text editor, Tim and Larry were pretty far along in designing a page layout system for graphic designers. They were still struggling with the issue of how to think about the user interface for documents and files, and the actions that take place on an entire document, rather than on the pages or text within a document. Tim describes his moment of inspiration:



Tim Mott's reconstruction of his sketch on the bar napkin

Photo
Author

> I was in a bar late one afternoon waiting for a friend, doodling on a bar napkin and thinking about this problem. I was just obsessed with this design at the time; I was just consumed by it. I was thinking about what happens in an office. Someone's got a document and they want to file it, so they walk over to the file cabinet and put it in the file cabinet; or if they want to make a copy of it, they walk over to the copier and they make a copy of it; or they want to throw it away, so they reach under their desk and throw it in the trash can.
>
> I'm sitting there thinking about this and I'm doodling. What ended up on the bar napkin was what Larry and I called the "Office Schematic." It was a set of icons for a file cabinet, and a copier, or a printer in this case, and a trash can. The metaphor was that entire documents could be grabbed by the mouse and moved around on the screen. We didn't think about it as a desktop, we thought about it as moving these documents around an office. They could be dropped into a file cabinet, or they could be dropped onto a printer, or they could be dropped into a trashcan.

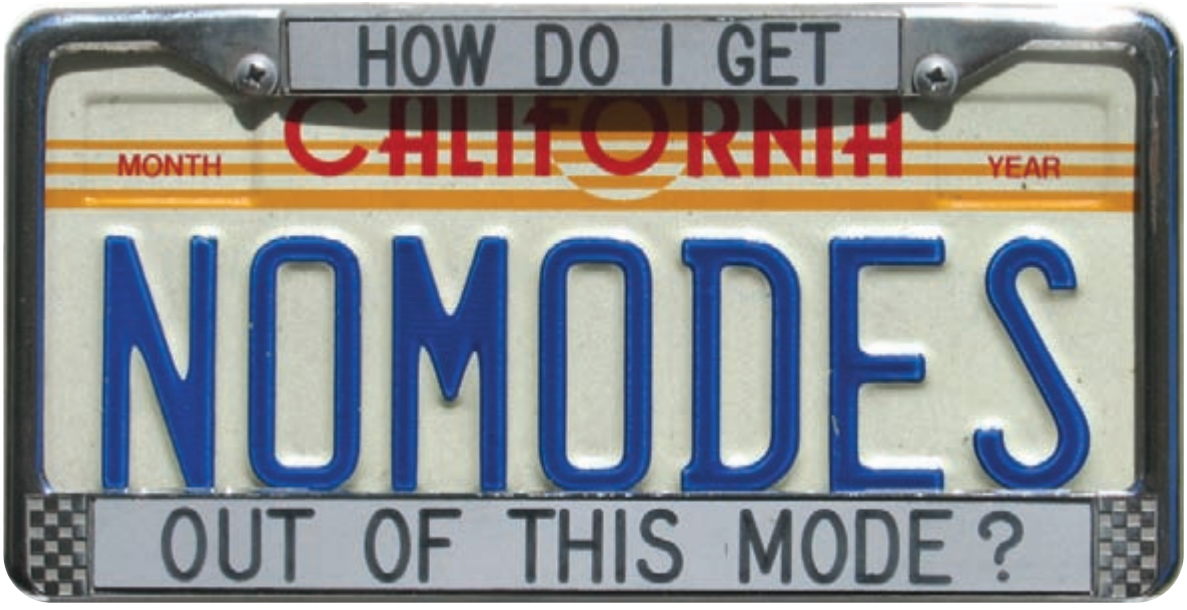
> The desktop was part of the design, and on it there were those things that you might normally find on a desktop like a calendar, a clock, and baskets for incoming and outgoing mail; the notion was that we would be able to use that as the controlling mechanism for electronic mail.

When Larry heard about the idea and saw the bar napkin, he showed Tim the illustrations from "Pygmalion,"[14] Dave Canfield Smith's thesis. He said that people had tried to implement similar designs before, but they had always attempted to do very complex, three dimensional, true-to-life simulations, as opposed to just a simple two-dimensional iconic representation. The simplicity of the representation was the breakthrough! Somewhere in one of Tim's notebooks in Xerox there is a bar napkin covered in the doodle of his office metaphor, complete with a desktop and trash can.

Photo Author

Larry Tesler

When he was at Apple, Larry Tesler had a license plate saying "NO MODES," emphasizing his passion for designing software that would be simple and easy to use. He had been writing code since he was in high school and worked at the computer center at Stanford while studying there in the early '60s. He founded his own company to offer programming services while he was in his junior year and soon discovered that his customers had a different way of thinking from the software engineers in the Computer Science Department. He realized that the best way to design the software was with participation from the customer, and he developed techniques for watching how people did things and designing software that allowed people to use new technology in familiar ways. He learned to create prototypes rapidly and to test them with the intended users early and often. After working at the Stanford Artificial Intelligence Laboratory, he went to Xerox PARC in 1973 in time to be a key player in the development of the desktop and desktop publishing. In 1980 he moved to Apple, where he was core to the design of Lisa. He invented cut-and-paste and editable dialog boxes, and he designed the Smalltalk browser. He simplified the use of the mouse by reducing the controls to a single button. He insists on truth and accuracy and is willing to challenge anybody's assumptions about the best way to do things, always thinking from basic principles. After a three-year stint at Amazon, running the usability and market research groups, he is now helping Yahoo improve the design of their online interactions, as VP of user experience and design.

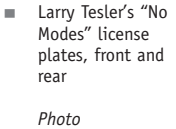


Larry Tesler

"Iconic naming systems will be explored. A picture of a room full of cabinets with drawers and file folders is one approach to a spatial filing system."

From a white paper called OGDEN (Overly General Display Environment for Non-programmers), by Larry Tesler and Jeff Rulifson

Participatory Design



Larry Tesler's "No Modes" license plates, front and rear

Photo
Author

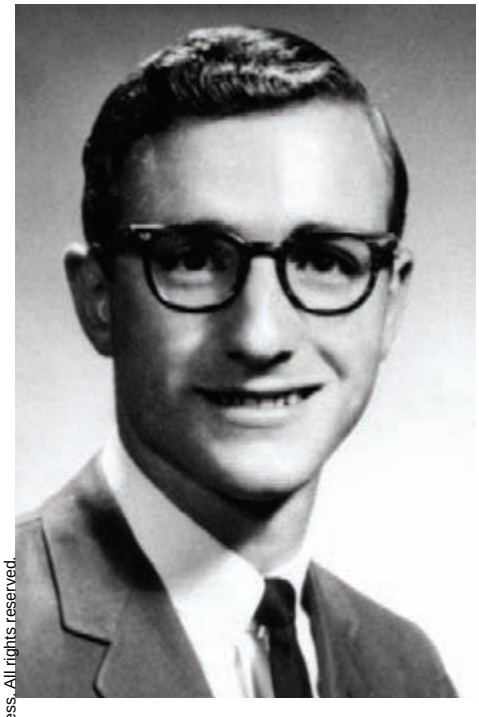
LARRY WAS QUICK to realize the value of participatory design and usability testing. His first lesson occurred while he was still studying at Stanford, when he was asked to get involved with a project to organize the "card stunts" at football games and to automate the production of instruction cards. At halftime during the football game, when the students flipped up the cards, instead of hand-written instructions they would be computer-generated. Students from the Computer Science Department had already written a programming language on punch cards, with all numeric commands. An art student would draw polygons to represent the shapes and colors and describe an animation, and then a computer science student would code all the instructions to generate the instructions and print out the cards. Finally the cards would be torn along perforations and set out on the seats. Larry was persuaded to take over the coding role.

"The trouble is that the art students have a great deal of difficulty with this language, so you end up coding all the stunts," he was warned; "if you don't want to code all the stunts, you better find a way of making it easier to use!"

He started talking with the art students and trying to figure out what they wanted. It was a different world! Over the next three years, he reworked this language several times, until even the art students could use it.

In 1963, when he started a software consulting business in parallel with his studies, he developed a rapport with users and worked out how to ask them the right questions, so that he could make programs they could use. He did not have to run the software for them; they could do it themselves. One of the programs he wrote was a room-scheduling program for the Stanford Psychiatry Department. They were trying to arrange for patients who came in to get assigned to rooms with doctors. They had scheduling problems because appointments overlapped, and rooms had to be used at certain times during the week for other things.

> One of the psychiatrists and I designed it together. That worked out really well, as they needed no help from me at all to use this. That's when I realized that the best way to design the software is with the customer, which is now called participatory design. With the art students and the card stunts, I would watch how they used it and see when they got confused. That was my first experience with what we call usability tests today, observing people and seeing what the problem was and then going and fixing it, and realizing that I could make something simpler. That was really the beginning of my interest in usability.



■ Larry Tesler at Stanford, 1961

The Future System Will Use Icons

Experience at the Stanford Artificial Intelligence (AI) Lab gave Larry lots of exposure to the computer science community in the San Francisco Bay area. He got interested in typography and developed a publication language called Pub. It was what today would be called a markup language with embedded tags and scripting, but this was in 1971. It only ran on the PDP 6, which limited it to use in universities. A lot of graduate students used it for their theses in the major universities that were on the ARPAnet. On request, he distributed it in source-code, so it was actually one of the first open-source markup languages.

He decided this was really the wrong way to do it, and that the right way to do it was to have an interactive layout system, not a language, so that anybody could use it. He was putting together a catalog for a volunteer group one day in 1970, using X-ACTO knives and glue, pasting up the pages with type-scripts and remembers saying to a friend:

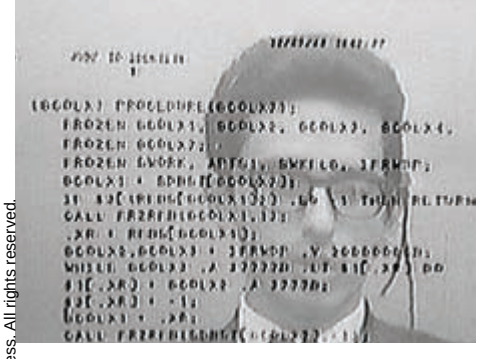
> You know, this is not the way this is going to be done in the future. In the future you're going to have a big screen in front of you, and you're going to be taking these excerpts from documents, and you're going to be pasting them into pages, without glue, and it'll be perfect. We won't have all these alignment problems! And then when you're done you're going to print on a phototypesetter. It'll all be done interactively.

He was looking for the right place and opportunity to turn this vision into reality, and in 1973 he joined Xerox PARC with that expectation in mind. When he got there, he said that he wanted to work on interactive document formatting. "We'll get to that, but first you need to work on this distributed office system we're building," Bill English replied. This was disappointing, but he was allowed to spend about half his time working with Alan Kay's team, which was much more exciting as they were starting to develop Smalltalk. He got involved there developing the first modeless editor.

Larry Tesler at Stanford AI Lab, 1971 ■
Larry Tesler at Xerox PARC, 1974 ■

Initially I started working with a guy named Jeff Rulifson, who had worked with Bill English and Doug Engelbart at SRI. "You two go off and do some visioning about the office system of the future," Bill said. That was pretty exciting, so we went off and wrote a little white paper on it called OGDEN (Overly General Display Environment for Non-programmers). A lot of it was kind of nuts and all wrong, but there were some really good things in there. Jeff had just read a book about semiotics, and he said that it had talked about signs and symbols and icons and so on. The author of the book had said that icons would be a really appropriate thing to use in an interface for a computer system, because they would remind people of a concept, and they would grasp the idea quickly by looking at the icon without having to read a word. Jeff thought that we should incorporate that, so we put it in our white paper, and said, "Iconic naming systems will be explored. A picture of a room full of cabinets with drawers and file folders is one approach to a spatial filing system."

We had a diagram with an idea of what it might look like. It didn't look at all like the Mac or Windows; it looked more like General Magic, with a 3D, or perspective, desk, a file cabinet in the corner, and things on the desk, and a trash can next to the desk. The idea was that you would point at stuff that looked semirealistic, representational, but not equal-sized icons like we have today. Alan Kay loved that idea and had been thinking of similar things, and he passed it on to Dave Canfield Smith when Dave came looking for a thesis to do, and Dave actually implemented the first thing that's more like what we think of as icons today.



■ Jeff Rulifson at Doug Engelbart's 1968 demo

The Five-Minute Learning Curve

With the exception of Jeff Rulifson and Larry, everybody at PARC was extremely wedded to the NLS interface that was developed in Engelbart's team. Jeff was the least wedded to it, perhaps because he had come up with it.

"Why is everybody so excited about this interface?" Larry said to Jeff, who laughed.

"I don't know, because it was never designed!" Jeff replied, "When they implemented the system, they didn't have a user interface; they just knew what they wanted it do."

Jeff kept asking what the user interface was going to be.

"Well, I haven't designed that yet," was the reply, "I'll do that later."

Jeff was assigned the task of testing the system, so he took the procedures in the language. There were commands like *delete*, *left parenthesis*, *character number*, *right parenthesis*, so he made something that was absolutely literally the same thing, only intended for the programmers testing the software. They all started using it and learned it, and then they started getting excited about how you could type the letters on the keypad by using binary code. What started as a testing language acquired a mystique; although it was never designed for end users and never analyzed scientifically, its aficionados grew to believe that it was an ideal interface for a system to augment the intellect. Perhaps it was. The trouble was that it was really hard to learn, and they admitted that.

Larry set out to prove that you could learn something in a day. He developed a little text-editing program that he called "Mini Mouse." It didn't use the NLS keypad at all and only used one button on the mouse. Before doing it he decided that he wanted to observe a user, and used a technique similar to his "guided fantasy." He describes working with a secretary who had just started at PARC and was not yet influenced by the programs that were being used:

> I sat her down in front of a screen, and did what's now called a "blank screen experiment."

"Imagine that there is a page on the screen, and all you've got is this device that you can use to move a cursor around, and you can type," I said. "You've got to make some changes to this document. How would you do it?"

I gave her a paper document with lots of markups on it for reference, and asked her to imagine that is was on the screen. She just designed it right there!

"I would point there, and then I would hit a delete key," she said.

To insert, she would point first and then start typing. She'd never been contaminated by any computer programs before, so I wrote all this down, and I thought, "That sounds like a pretty good way to do it!" Later I also tested other new people like that and got similar kinds of results. I wrote a report about this. Up to that time, Bill English had been extremely skeptical about what I was doing. He seemed offended that I was challenging the "perfect" NLS editing language, but when he read my document a light went off, and he came to me and said, "This is really good!"

In the past, the programmers at SRI and PARC had thought that usability tests would be extremely expensive and time consuming and had done very few of them. Larry felt that he had to demonstrate that testing did not have to be difficult or expensive, so he built Mini Mouse in Smalltalk, like a typewriter. It turned out to be so easy to use that they could take people who were walking by in the building, and ask them to try it, and they were able to learn to operate it in five minutes—compared with a week to get comfortable doing comparable tasks using NLS.

Double-click, Cut, Paste, and Cursors

The Mini Mouse triumph earned Larry the freedom to work more independently and to focus on text-editing software. When the gauntlet was thrown down by Ginn to help them with a publishing system, he was only too pleased to take on the task. He found Tim Mott, and they became partners in creating a new state

of the art for text editing and graphic layout, as described in Tim Mott's interview. Larry tells the story of inventing the double-click:

> When I had written up Miki Mouse, which was going to be the next thing after Mini Mouse, I was trying to decide what to do with all this hardware. We had three buttons on the mouse. I really wanted it to be a single-button mouse, because I wanted to be able to use other things like tablets, touch screens and light pens, and you couldn't do that with a multibutton design. Another reason was that when people were using the software, they thought of the mouse as being held by the pointing hand, with the other hand being for commands, either on the keyboard or on the five-key keypad. I thought if we could separate that out really cleanly we would reduce the common mistake of hitting the wrong button on the mouse. One in five times it was, "Oops, hit the wrong button." This was why you had to practice a lot to master NLS, as so often you hit the wrong button.
>
> "Maybe we could use the first button to select the cursor location between characters," I thought, "and the second button to select a word, and the third to select a sentence or something." I was with Tim and said, "I really don't like this, because I'd really rather just have one button."
>
> One morning he came in and said, "I've got it; double-click! You click twice in rapid succession in the same place to get a word, and three times to get a sentence."
>
> "Twice maybe, but not three times, surely!" I said. I gave him all the reasons that it wouldn't work, but when I closed my eyes and tried to envision it: "Yes it just feels right," I had to admit, "double-click to select a word; it just feels right."
>
> We had already been implementing Gypsy at that point, and some of it was working enough so that we could try it. We brought in the secretaries and had them try it, and everybody thought it was good. There were lots of detailed technical arguments from the programmers, but basically it became a good thing right away.

One day Larry saw a novel way of handling "delete" and "insert" that had been developed by Pentti Kanerva, who was a software expert at Stanford University. It was done by first selecting what you wanted to delete and then saying *escape-d-*

return for delete, and then by moving the cursor where you wanted it and then saying *escape-o* for "oops," which meant "undo last delete." If you made a mistake, you just hit L, which meant insert, and it inserted whatever was in the buffer wherever the cursor was. The cursor was likely to be where it was before, so it came back.

The magic of this idea was that if you moved the cursor somewhere else after you said delete, it would move it there, so you didn't need a *move* command. When Larry was working on Gypsy, he remembered that and thought that he could use it to avoid a special mode for moving. The trouble was that it was not good for copying, because you would need to copy it to the buffer.

> We were doing this for a publisher, Ginn and Company, and we were planning to implement "cut" and "paste" to move things around in the document.
>
> "Why don't we use the same words, *cut* and *paste*, for this process of delete and insert," I said to Tim, "and that way if you want to copy, you can say *copy* and *paste*."
>
> Tim agreed, so we named our commands, *cut/paste* and *copy/paste*. Because of that, I seem to have the reputation for originating cut-and-paste, but it was really Pentti Kanerva's idea that I renamed and reduced the number of keystrokes from six to two.

■ The original "cut" and "paste" labels written by Larry, stuck onto the NLS keyset to first test the idea

Photo
Nicolas Zurcher

There was no cursor in NLS: you just pointed at things and they became the first item (character, word, and so forth) or the last item of the selection. Every system that had a cursor took a character and either underlined it or showed it in reverse video; normally if you were doing white on green, you'd show green on white. The problem with that was that if you moved the cursor somewhere and started typing, it wasn't clear where the character would go. The way it worked in most systems was that the character that you had inverted would change as you typed, so you would overtype things. If you wanted to insert, you had to go into a mode, like control-i or an insert key, and then everything you typed would get inserted before the character. Larry found in experiments that a lot of people expected it to go after the

character, and some people were just confused. He didn't want something you had to learn; he wanted it to be obvious. He was puzzling about this, and brought it up it during a meeting at PARC.

A few days later he bumped into Peter Deutsch in the hallway, who was not a user interface guy in any way; he was more of a systems guy.

"I think I have a solution to your problem of insertion for the cursor," he said. "Put the cursor between characters, not on a character."

"That's a nice idea, but how do you know which character you mean?" Larry said.

"Well, if you click on the right half of the character, the cursor goes after it, and if you're on the left half of the character, it goes before it."

"Oh, of course, that's so simple. It's the answer." Larry thought, "I just need a way to show that there's this place between characters. In this publishing system they're used to using this caret mark to show that something needs to be inserted, so I can use that."

He started playing around with the caret mark, and finally came up with a caret that would work, overcoming the space management problem, and in the early systems they used a little caret mark as the symbol. Later, Dan Ingalls came up with the idea of just doing a vertical line between characters and implemented that for Smalltalk. They made it easier to see by making it blink on and off, to avoid confusion with a vertical bar. Larry tried other alternatives to make it more visible, for example curling the top and bottom like a sideways H, so that it didn't obscure the character you were looking at, was easy to find on the screen, and had a clear meaning. The vertical line was the design that survived.

Smalltalk Browser

LARRY HAD BEEN trying to get into Alan Kay's group from the moment he arrived at PARC. As a result of the success of the Gypsy project, he was given the chance, and he started working on Smalltalk. He describes his delight in working for Alan and how he was challenged to design a browser:

> Alan was always a great research manager; he would always challenge us, and would never be satisfied with anything. He was very good at rewarding people and acknowledging their work.
>
> He would say things like, "We still don't have a good way to query databases," or "We still don't have a good way to deal with animation."
>
> One of his favorites was, "You know, you can go to a library and you can browse the shelves, but we don't have any good way to browse."
>
> Diana Merry, who had converted from being a secretary and become a Smalltalk programmer, said to me one day, "You know, we still don't have a good way to browse."
>
> "I'm so tired of hearing that, Diana," I said, "Alan says that every month, and I'm so tired of it! It isn't such a hard problem."
>
> "Well then, go do it!" she said.
>
> "If I do it, will you promise never to say this again?"
>
> She assured me that she wouldn't, so I had to figure out how to browse. I was thinking of graphical metaphors with books on bookshelves, and then thought that maybe I should start with something closer to home. We programmers are always trying to understand other people's code, scrolling through big files of source code all the time trying to find stuff, and we don't know what it's called, so we can't use the search command. I thought that if I could figure out a way to browse through code, we could maybe generalize it to work for other things too, plus it would be a cool thing to have for myself, so I decided to build a code browser.
>
> I think this is one of those things that a lot of designers do. You just kind of close your eyes and vision, what would it look like? Instead of putting the secretaries in front of a blank screen, for a programmer's tool, I'm designing for myself, so I can be the subject.

Close your eyes. Blank screen: what does it look like? Smalltalk had classes of objects, and methods, which were routines or procedures.

"Okay, I'll have a text list of classes, and then when I click one of them, somewhere it'll show all the methods," I thought. "Then when I click the method, it'll show me the code for that method. Then I can scroll through the lists, and browse around. That's pretty simple!"

I got a piece of paper and drew it out and had the usual width problems with lack of room for three columns, so I put the two lists of classes and methods across the top, to allow a full width window for the code. It was a window broken up into three pieces, so I'll call them "panes," so we can have windows and panes. Each pane will have its own scroll bar so you can browse before you pick one to see the code. While we're at it, once you're looking at the code, I might as well let you edit it, using the standard Smalltalk editing facilities of drag through, cut-and-paste and all. I built it in somewhere between one and two weeks; it was one of those "Aha" kind of things, and it immediately started getting used by everyone. There were over a hundred classes and over a hundred methods; Dan Ingalls made a big improvement to it by dividing them into categories, so we ended up with four panes across the top. That became the classic Smalltalk browser.

This was the first browser design. It was a welcome change for Larry to design something for himself as a software engineer. He had become so wedded to the ideas of participatory design and user testing, that it seemed surprisingly straightforward to be designing for the user in the mirror. He went on to use the Smalltalk browser to build a point-and-click debugger and inspector. Other people have used the same metaphor to browse around databases. Although Web browsers are fundamentally more like Engelbart's NLS, which used links to jump between documents, frames in Web browsers resemble Larry Tesler's browser panes from 1976.

The Brain Drain from PARC

Around 1980, lots of people started leaving Xerox PARC.[15] At the beginning, when PARC was founded, the most inventive minds in technology had been attracted by the promise of creating something that would change the world and by the chance to work with and learn from each other. They had been together long enough to be strengthened by their collective experience, and many of them were itching for the opportunity to see their ideas move from research to reality. Xerox was struggling to maintain dominance in the copier market, as Japanese competitors were overtaking them at the lower end of the market and gradually moving up. The only part of the research at PARC that Xerox really took advantage of was the laser printing technology, and although that produced enough revenue to justify the investment in the research center many times over, it was increasingly clear to the people there that concepts for the "office of the future" were on hold. There was also a sense of opportunity in Silicon Valley, as the venture capital firms were starting to fund new-start companies for the development of products rather than just for chips. It was natural for all those brilliant researchers to want to take the next step in their careers, so they were looking around.

As it happened, Larry Tesler, Alan Kay, and Doug Fairbairn all resigned from Xerox on the same day, for three different reasons, to go three different places, without knowing one another's plans. Larry was headed for Apple, where he started in July 1980. He had participated in the famous demo of PARC technology to Steve Jobs and the team from Apple and was getting to know more about them:

> I was blown away by the Apple people. I had been trying to convince people at Xerox about my belief in the future of personal computers, but the Apple people appreciated everything in the demo that I thought was important. I agreed with all the things that they said should be done next.
>
> "They think more like me than the PARC people do." I felt, "I'm in the wrong place! I should join a company like Apple."

Two years before I joined Apple, when the company had only thirty employees, a friend had dragged me along to a company picnic, but they still seemed to me like a bunch of hobbyists. Between the picnic and the demo, they had hired all these computer scientists and software engineers who were very sophisticated.

"These guys really know what they're doing," I thought, "and they want to do the right thing!"

When I got to Apple, I said once again, "What I want to work on is publishing systems," but they asked me first to work on Lisa. They put me in the research group consisting of three people: Bruce Daniels, who was a systems guy; David Hough, who was a numerical analyst; and me.

"I don't really understand what kind of research we're going to do," I said after I had been there a week. "We don't have time to do research. Apple is not ready to do research. It's too early in the history of the company."

They agreed, so we went to the management.

"We've just dissolved our research group," we said, "so you should just put us in the Lisa group."

They did, so I started talking to Bill Atkinson about what I thought should be different. Bill was a neuro-psychologist by training, with a great combination of psychology, design (he's now a photographer), and programming. He was a great integrator; he was a sponge. At first I didn't think he was hearing people's ideas, but he was processing them and synthesizing them in a wonderful way. It would be great to be inside Bill Atkinson's brain!

This was the start of a great collaboration between Larry Tesler and Bill Atkinson. They formed a tight bond and creative partnership and went on to design some of the most significant and long-lasting interactions in the world of desktops and windows. An interview with Bill Atkinson follows in chapter 2, and he and Larry tell some of these stories together.