Chapter 3
The Two-Eyed Man

Alan Kay

I was unable to be at Chapman University for the celebration of Ted Nelson's life's work, so Bonnie MacBird and I made this video to celebrate your day.[1] We wanted to thank Ted for his role in us meeting up, falling in love, and getting married. In the video, Bonnie explains:

> This is how Ted Nelson spawned the movie Tron and a marriage that's lasted more than thirty years. The year was 1979. I'd just left Universal Studios to write a movie about a video game warrior inside of a computer. There were no personal computers at that time [beyond] these. In L.A., there were many video arcades, but only one computer store for Home Brew-types only. I went there and found this book: *Computer Lib/Dream Machines* by Ted Nelson (Fig. 3.1).
>
> I read it cover to cover. Well, cover to middle, then upside down, and other cover to middle. There was an article about Alan Kay, so I went up to Xerox PARC and met the guy. A half hour meeting stretched into hours and Alan Kay became the technical consultant on the movie *Tron*. We spent many happy hours in conversation along Venice and Santa Monica beaches. I wrote a script filled with "cool" science. There was a bit who wanted to be a program, and there was a video game warrior who wanted to be a human. The script was uploaded to PARC, and then I went up there and edited the script on the Alto computer, making *Tron* the first movie script ever to be edited using a Word Processing program. It sold to Disney and after eight new writers and considerable meddling it became the movie *Tron*. Groundbreaking, yes, but Alan and I think the marriage turned out better than the movie! We thank you, Ted Nelson.

As Thorton Wilder's old fortune-teller says, "It is easy to tell the future," but asks "who can tell the past?" It's not just a memory problem, but one of too much complicated detail without enough perspective. It would be great if we could go back and look at the world Bonnie talks about, and, to some extent, we can.

[1] This chapter has been transcribed and edited from a video created for the Intertwingled conference (Alan Kay Talk at Ted Nelson Tribute: https://www.youtube.com/watch?v=AnrlSqtpOkw).

A. Kay (✉)
Viewpoints Research Institute, Los Angeles, CA, USA
e-mail: alan@vpri.org

D.R. Dechow, D.C. Struppa (eds.), *Intertwingled*, History of Computing,
DOI 10.1007/978-3-319-16925-5_3

Fig. 3.1 Bonnie MacBird with a copy of Ted Nelson's *Computer Lib/Dream Machines*

Some years ago, Xerox decided to clean its warehouse and throw out most of the PARC data disks.

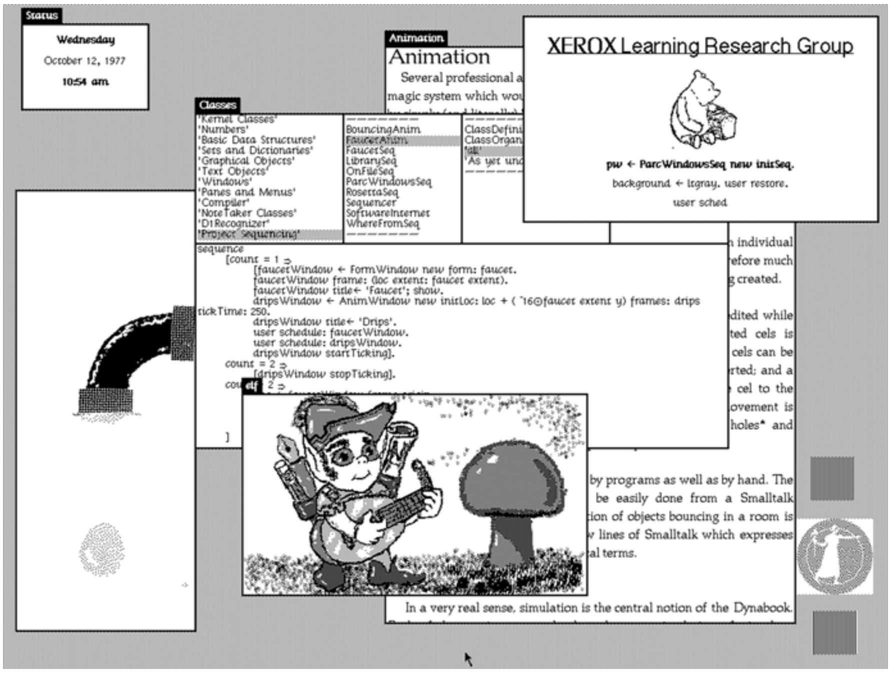
Roughly one hundred disks out of thousands were rescued, and a few thousand files were recovered. A single one of all those files happened to contain one of our systems from the 1970s (Fig. 3.2). Smalltalk was completely self-contained. There's no separate operating system, applications, etc., only software computers communicating with each other and each simulating some aspect of the personal computer system. Some objects simulate characters on the screen; some simulate pictures; some, windows; some, places where the users can do things.

The software computers are, in terms of virtual hardware, independent of the physical computers they run on. To bring this back to life, we emulated the virtual hardware in Javascript. It is faster than the actual PARC computers of 40 years ago! With this approach, we have a time machine that allows us to go back, back, back into the past and run the same software that both Bonnie and Steve Jobs saw.

In Fig. 3.3, we see familiar forms: overlapping windows, iconic representations, and so forth. Windows are objects that are *views* of objects: tools and the kinds of resources that media authors use to create the writings of the future. They're not stovepiped "apps." You can bring any and all objects in the Smalltalk system to any of these projects. For example, here we see a view of the system itself and animation. A half-tone painting I did 40 years ago. I can scribble it up a little bit for you. Here's some text. This system also had a gesture recognizer.

Now let's go to the project where I organized this talk. In Fig. 3.4, we see many small windows that look unto projects of their own. We can think of this system as having unlimited "desktops" on which "projects" can be done, and all the resources needed for each project can be brought there and they'll persist over time.

Fig. 3.2 A hard drive (location shown in *circle*) containing a Smalltalk image from the 1970s was retrieved from the digital trash heap



Animation
Several professional a
magic system which wou

```
'Kernel Classes'             --------          --------
'Numbers'                    BouncingAnim      ClassDefini
'Basic Data Structures'      FaucetAnim        ClassOrgan
'Sets and Dictionaries'      FaucetSeq         'all'
'Graphical Objects'          LibrarySeq        'As yet unc
'Text Objects'               OnFileSeq         --------
'Windows'                    ParcWindowsSeq
'Panes and Menus'            RosettaSeq
'Compiler'                   Sequencer
'NoteTaker Classes'          SoftwareInternet
'D1Recognizer'               WhereFromSeq
'Project Sequencing'         --------
```

```
sequence
    [count = 1 ⇒
        [faucetWindow ← FormWindow new form: faucet.
        faucetWindow frame: (loc extent: faucet extent).
        faucetWindow title← 'Faucet'; show.
        dripsWindow ← AnimWindow new initLoc: loc + ( ¯16⊙faucet extent y) frames: drips
tickTime: 250.
        dripsWindow title← 'Drips'.
        user schedule: faucetWindow.
        user schedule: dripsWindow.
        dripsWindow startTicking].
    count = 2 ⇒
        [dripsWindow stopTicking].
    cou   2 ⇒

    ]
```

```
pw ← ParcWindowsSeq new initSeq.
background ← ltgray. user restore.
user sched
```

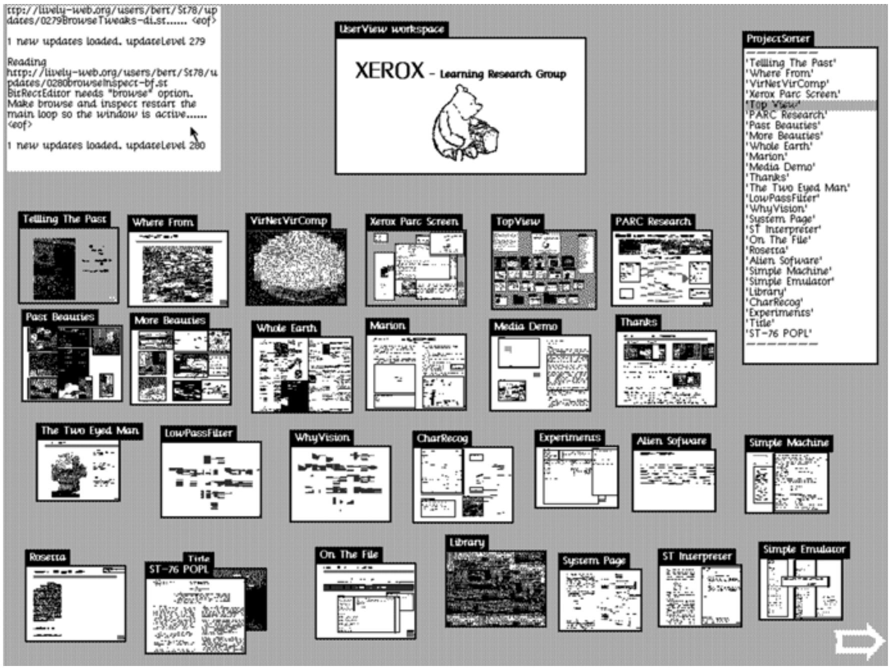
n individual
efore much
g created.

dited while
ted cels is
cels can be
rted; and a
cel to the
ovement is
holes* and

by programs as well as by hand. The
be easily done from a Smalltalk
tion of objects bouncing in a room is
v lines of Smalltalk which expresses
al terms.

In a very real sense, simulation is the central notion of the Dynabook.

Fig. 3.3 Image of a Smalltalk screen at Xerox PARC in the 1970s



```
ttp://lively-web.org/users/bert/St78/up
dates/0279BrowseTweaks-di.st...... <eof>

1 new updates loaded. updateLevel 279

Reading
http://lively-web.org/users/bert/St78/u
pdates/0280browseInspect-bf.st
BitRectEditor needs "browse" option.
Make browse and inspect restart the
main loop so the window is active......
<eof>

1 new updates loaded. updateLevel 280
```

Fig. 3.4 Screen showing many project windows. They can be arbitrarily linked

Anything can be done in each of them. They can be linked together in any way; they are not hierarchical. I'll enter one—a typical media screen (shown in Fig. 3.5)—that describes PARC.

This work was part of the "elephant of personal computing," which, as in the fable of the blind philosophers, is interpreted in different ways by different researchers.

The ARPA IPTO (Advanced Projects Research Agency Information Processing Techniques Office) community had lots of different views. The basic idea of ARPA was to avoid the disputes over different points of view that were part of the blind philosopher's fable and try to do what scientists have done to figure out a universe that we can only approach piecemeal. PARC was an offshoot and microcosm of this community starting in the 1970s, and individual researchers were often part of more than one research area. I was part of the Learning Research Group.

Another group was the Computer Systems Lab, which did much of the hardware heavy-lifting and day-to-day tools. One group that is less well-known is the POLOS Group (PARC OnLine Office Systems), which was made from some of people who came over to PARC in the early 1970s from Doug Engelbart's group.

A myth about PARC was its extreme originality. One of the triumphs of a few hundred years ago was to be able to make globes of the earth as if it would look if we were out in space. Two hundred years later, the views in the 1980s were quite

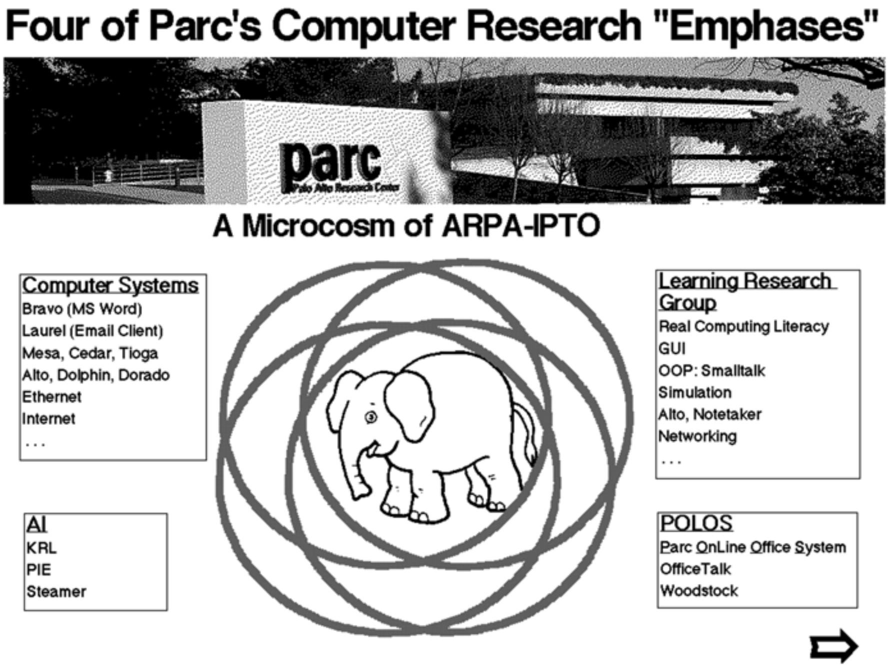


Fig. 3.5 Some of the many facets of the invention of personal computing at Xerox PARC

identical to the globes of 1780. There were hardly any surprises. Likewise, it is, perhaps, more accurate to claim that we in PARC were less original in the 1970s than we had been in the 1960s when many of the ideas were invented and explored for the first time (Fig. 3.6).

In the early 1960s, there was an enormous wealth of ways to think about personal computing and networks, including Sketchpad, the very image of personal computing. Some of the personal computing explorers included Douglas Engelbart, of course, and Ted Nelson and Andy van Dam. The Grail Gesture Recognition System on a tablet that was invented the same year as the mouse—1964—and the conventions of making arrows, windows, and so on, including moving and resizing them. All of this was happening at that time: Seymour Papert with his Logo programming language and Turtle graphics; Simula; and some of our own stuff as well, such as the Arpanet, the Flex Machine and its first object-oriented operating system, the idea of Dynabook, and much, much more. It was an exciting time.

The *Whole Earth Catalog* and its folks were nearby in Menlo Park thinking big thoughts about universal access to tools. Not just physical, but especially mental. This was the first book in the PARC library, and it had a big influence on how we thought things should be. We loved the idea of lots of different tools being available with explanations and comments, and we could see that it would be just wonderful if such media could be brought to life as one found and made it. This thinking led to

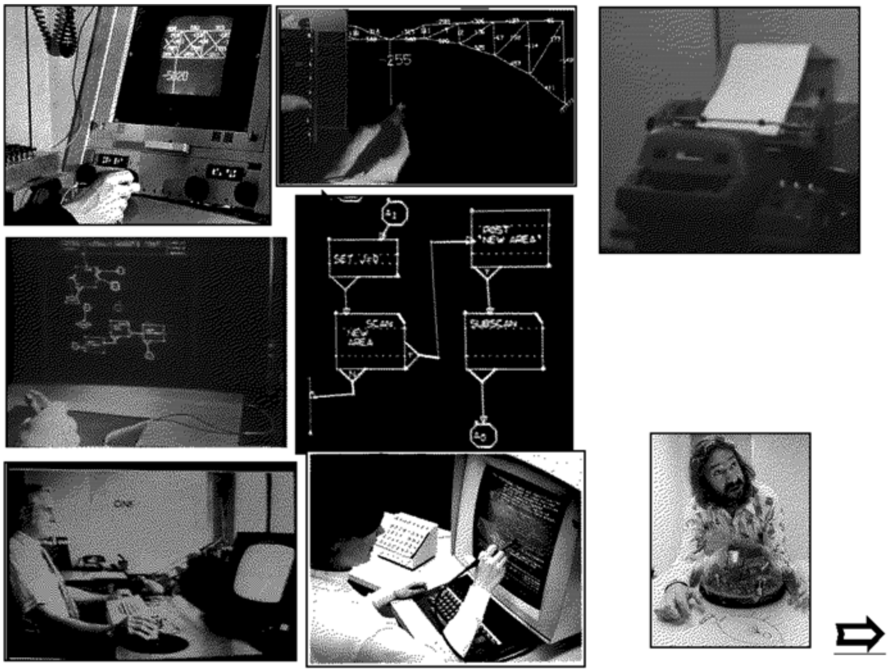


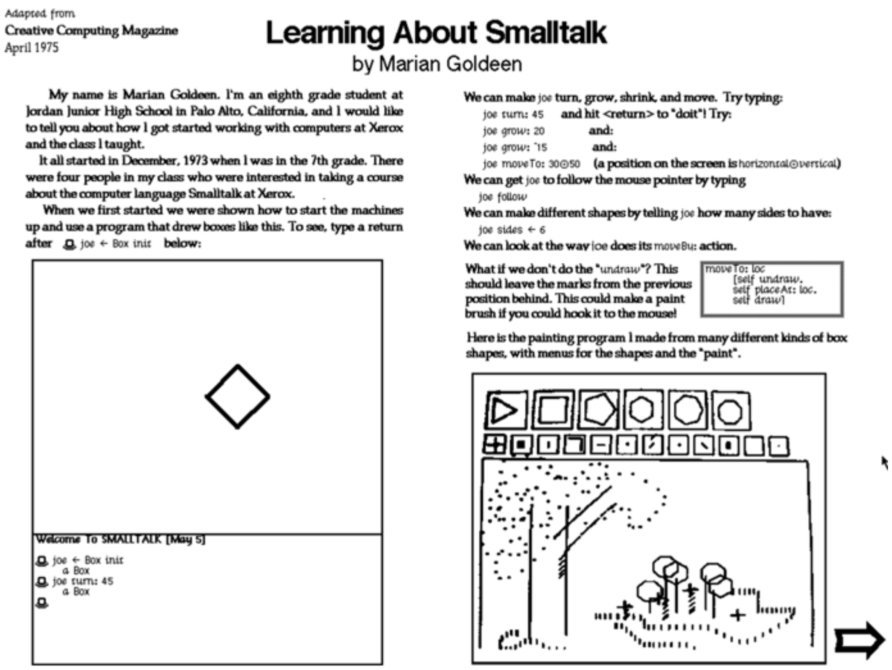
Fig. 3.6 Some of the precursors of the work at Xerox PARC

ideas of how to explain and explore by actually making things from computer stuff in the kind of general literacy we have for reading and writing. Now, we could include the reading and writing of dynamic models. This kind of literacy is best learned by children, so we started to work with them.

In my conference presentation, I showed the computer version of an article that 13-year-old Marian Goldeen wrote in *Creative Computing Magazine* in 1975 about what she'd done the previous year in our group. The computer version goes beyond reading to allow the reader to try out the very things that Marian is talking about. We called this form an active essay (Fig. 3.7).

In the middle of the essay is a simulation of the Alto screen so one can see what things looked like in her projects and do the same things that she did. She started off by making a box object called Joe that can be sent messages to get it to behave. Programming in Smalltalk is more like training intelligent agents than it is like the more standard metaphor of a cook making something from inert ingredients.

I showed a demo we used to do that combined animation and painting tools. The animation effect depends on what the brain does when it sees two different images, one right after the other. Animators like to say that animation takes place in between the frames. This means that we'd really like to do the redrawing of the bottom frame while the animation is running. But these are different tools. In the demo, the animation tool is animating the bouncing ball, and you can see that it's a bit weak. We'd expect that the ball would deform when it hits the ground.



Adapted from
Creative Computing Magazine
April 1975

Learning About Smalltalk

by Marian Goldeen

My name is Marian Goldeen. I'm an eighth grade student at Jordan Junior High School in Palo Alto, California, and I would like to tell you about how I got started working with computers at Xerox and the class I taught.

It all started in December, 1973 when I was in the 7th grade. There were four people in my class who were interested in taking a course about the computer language Smalltalk at Xerox.

When we first started we were shown how to start the machines up and use a program that drew boxes like this. To see, type a return after joe ← Box init below:

```
Welcome To SMALLTALK [May 5]
joe ← Box init
   a Box
joe turn: 45
   a Box
```

We can make joe turn, grow, shrink, and move. Try typing:

```
joe turn: 45       and hit <return> to "doit"! Try:
joe grow: 20       and:
joe grow: ¯15      and:
joe moveTo: 30⊙50  (a position on the screen is horizontal⊙vertical)
```

We can get joe to follow the mouse pointer by typing

```
joe follow
```

We can make different shapes by telling joe how many sides to have:

```
joe sides ← 6
```

We can look at the way joe does its moveBy: action.

```
moveTo: loc
    [self undraw.
    self placeAt: loc.
    self draw]
```

What if we don't do the "undraw"? This should leave the marks from the previous position behind. This could make a paint brush if you could hook it to the mouse!

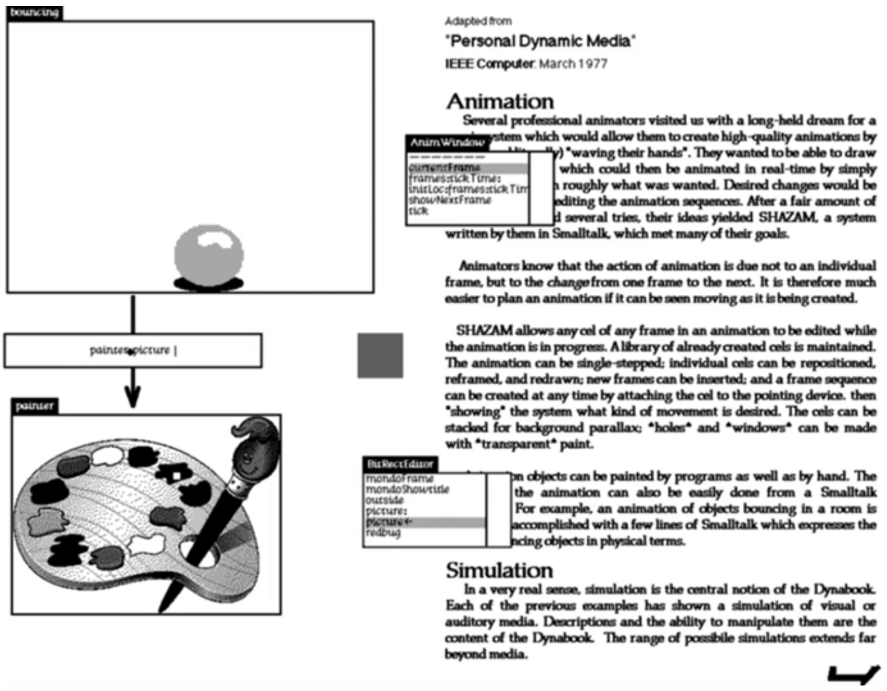
Here is the painting program I made from many different kinds of box shapes, with menus for the shapes and the "paint".

Fig. 3.7 Image of Marion Goldeen's "active essay" article for *Creative Computing* magazine

If there were apps in a commercial version of personal computing, we'd most likely expect that they don't talk to each other and it would be difficult to get them to talk to each other. This is a pet peeve of Ted's. But in this animation, they are just objects, and any object can talk to any object.

I'll take a look at the menu of the animation window. We can stop it ticking, and we can single-step the frame we care about. Maybe we want to share this frame with a painting tool. If this was prepared ahead of time, it would already be done. Instead, to paraphrase Thoreau, we need to find out what Texas might have to say to Massachusetts; that is, how did each of the tools characterize their parts and behaviors. Then we can do what Ted loves, shown in Fig. 3.8, which is to draw a line between the two windows. Some of the actions can be pre-defined, but we can also define one later by doing this gesture to create a dynamic link between the two windows. The painter's picture wants to be linked to the bouncing window's current frame, so we just write that in there and do it. The animation can be started again, and I can start painting the deformed ball. In the end, it starts to look pretty good. To prepare for the conference presentation, we had a terrific time bringing this old system back to life over the previous few months.

All the demos and forms I used in my talk were derived from old examples shown and published in the 1970s and made without changing Smalltalk's graphic system. The beautiful one-bit pictures use the Floyd-Steinberg technique, which



Adapted from

"Personal Dynamic Media"

IEEE Computer: March 1977

Animation

Several professional animators visited us with a long-held dream for a stem which would allow them to create high-quality animations by ly) "waving their hands". They wanted to be able to draw which could then be animated in real-time by simply roughly what was wanted. Desired changes would be editing the animation sequences. After a fair amount of d several tries, their ideas yielded SHAZAM, a system written by them in Smalltalk, which met many of their goals.

Animators know that the action of animation is due not to an individual frame, but to the *change* from one frame to the next. It is therefore much easier to plan an animation if it can be seen moving as it is being created.

SHAZAM allows any cel of any frame in an animation to be edited while the animation is in progress. A library of already created cels is maintained. The animation can be single-stepped; individual cels can be repositioned, reframed, and redrawn; new frames can be inserted; and a frame sequence can be created at any time by attaching the cel to the pointing device. then "showing" the system what kind of movement is desired. The cels can be stacked for background parallax; "holes" and "windows" can be made with "transparent" paint.

on objects can be painted by programs as well as by hand. The the animation can also be easily done from a Smalltalk For example, an animation of objects bouncing in a room is accomplished with a few lines of Smalltalk which expresses the ncing objects in physical terms.

Simulation

In a very real sense, simulation is the central notion of the Dynabook. Each of the previous examples has shown a simulation of visual or auditory media. Descriptions and the ability to manipulate them are the content of the Dynabook. The range of possibile simulations extends far beyond media.

Fig. 3.8 Smalltalk demonstration of a dynamic connection between two windows

was worked out at Stanford and PARC at the same time our system was built. But back then, we hardly used pictures like these, or many bit-map paintings because there simply wasn't enough storage to hold them. It's nice to take advantage of the larger storage capacities today. An iPhone's storage, for example, is many tens of thousands times larger and faster than the PARC machines.

An ancient proverb says that, in the country of the blind, the one-eyed man is king. Robert Heinlein's version of this proverb is that, in the country of the blind, the one-eyed man is in for one hell of a rough time! My version is that, in the country of blind, the one-eyed people run things and the two-eyed people are in for one hell of a rough time. That said, we owe much of civilization to the insights and suffering of the tiny number of two-eyed people. Ted Nelson was one of those few two-eyed people. We owe much to him, and this is being celebrated in this collection of essays.

A two-eyed person—Ted Nelson—comes up with a glorious symphony of how life will be so much deeper and richer if we just did *X*, but the regular world acts as a low-pass filter on the ideas. In the end, he is lucky to get a dial tone. The blind won't see it, and the one-eyed people will only catch a glimpse, but all of them think their sense or glimpse of the elephant is the whole thing. In our day and age, if they think money can be made from their glimpse, something will happen. They want to sell to the mass market of the blind so they will narrow the glimpse down even more. They could be educators and help the blind learn how to see; this is what science has done for the entire human race. But learning to see is a chore, so most, especially marketing people, are not interested. This is too bad, especially when we consider the efforts the two-eyed people like Ted have to go through to even have a glimpse happen. One of the keys is for the two-eyed people to turn into evangelists. Both Ted and our mutual hero, Douglas Engelbart, worked tirelessly over their lifetimes to point out that, in this dial-tone world, the emperor not only has no clothes but his cell phone can't transmit real music. Yes, I've mixed a metaphor or two.

Another key is to make a working system of the future. This was ARPA's and especially PARC's main mission. Make something that works, not just for a demo, but for a group of people. Some of what I showed during my talk is what Steve Jobs saw, and the Macintosh was a result of his glimpse and also interpretations of that glimpse by him and others at Apple. But it missed a number of really important ideas. Many of Ted's and Doug's ideas have been missed.

So, with all this working against someone like Ted, why bother having visions? Standard schooling is already trying to convert two-eyed children into standard children, that is, into blind children. Why not just put more effort into this and save all the bother?

To me, the visionaries are the most important people we have because it is only by comparing their ideas with our normal ideas that we can gauge how we are doing. Otherwise, as it is for most people, normal becomes reality, and they only measure from that less broad view of reality. Toss Ted back into this mix, and you've upset the Apple cart—and that's what we need! This allows us to see that normal is only one of many possible constructions of reality, and some of them could have been much better. In addition, the normal ideas in the future could be very different and much better from what is considered reality today.

Let's be very thankful that we live in a place where two-eyed people were really supported in the 1960s and at least tolerated today. And let us also be thankful that we have a two-eyed person like Ted Nelson who has been tirelessly energetic about not just having ideas but also about going out and telling people about those ideas, not letting them die, not letting them get absorbed into the low-pass filter.