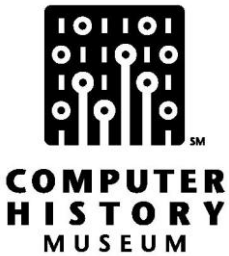

Oral History of Alan Kay

Interviewed by:
Dag Spicer

Recorded: November 5, 2008
Mountain View, California

CHM Reference number: X5062.2009

Dag Spicer: We're here today, November 5, 2008, with Dr. Alan Kay. We're at the Computer History Museum in Mountain View, California, and Alan has kindly agreed to do a little directed interview with us today, and welcome.

Alan Kay: Thank you.

Spicer: Happy to have you here. Alan, I wanted to start with a high-level question, which is why do you think computer history is important? Why should we look at what's come before?

Kay: Well, of course, there's the famous idea that people who don't pay attention to history are condemned to repeat it, and unless they're as smart as the original people who made it, they might even do worse. So in the old days people were admonished for reinventing the wheel, but today I think most of the old-timers would love for the new people on the scene to at least reinvent the wheel but it seems more like they're reinventing the flat tire. And I tell a lot of young people that you can easily be smarter than any single person back 30, 40, 50 years ago, but there's no way you can be smarter than a hundred of them. And there were a hundred people who were funded better than anybody has ever been funded in the last 25 years with a large set of visions that are almost inconceivable in the commercial world that computing has become and the combinationof all of those things, and also the fact that there is less noise, that because you couldn't make a lot of money the people who did it were strongly attracted emotionally to it. And a very large percentage of those people were super smart and super knowledgeable in other contexts that they could use as analogies for what computing might be like. And so I think the combination of all of those factors makes a typical pass through say starting 50 years ago when McCarthy invented LISP, and that's probably the anniversary that should be celebrated this year at the Computer Museum. I can't believe that nobody noticed. It has been celebrated a little bit out in the language world but that was probably the most significant thing ever done in programming languages, and the other things that McCarthy did around 1958, right on either side of it, were…probably three of them are in the top ten of all time. And so- but there's still time left in the year to—

Spicer: To mark that—

Kay: --get John down and talk to him. So the thing that I think that the- that most people especially in a pop culture, which is kind of what we have now, can't quite work their head around was…is the idea that there's nothing inevitable about progress, but because pop cultures don't know about history, they don't know of things about the Dark Ages when for a thousand years people forgot what the Greeks especially and the Romans put together and so there's this kind of false notion of Darwinian evolution must be optimizing. And any biologist will tell you it doesn't optimize. It just fits to whatever the environment is. If you have a weak environment, you're going to get weak fits to it. So I think all of those things together say that looking into the past at the best stuff, and of course there was a lot of bad stuff done back then, looking at the best stuff I think would be very enlightening to many people. On the other hand, our field is somewhat at fault because we haven't done what the molecular biologists have done. They made a book called “The Molecular Biology of the Cell,” [Alberts, et al] which is a masterpiece, 1,200 pages, starting off requiring no knowledge of chemistry even; 1,200 pages later beautifully exposed is, you more or less understand what molecular biology is all about. In computing it's more difficult because we have a synthetic field rather than analytic. We aren't given a universe full of phenomena so we create the phenomena so that means we're basically a design field and a style field whereas nature picked DNA a long time ago and DNA has successfully held off everything else. And so you get a kind of a continuity of style in molecular biology that allows you to pick 20 things and then hang the whole deal on, but I think still today you could pick one of the successful styles. And of course I would suggest the style of the ARPA IPTO and Xerox PARC style because things like personal computing and GUIs and the Internet, Ethernet, that style gave rise to some of the most powerful technologies we have today. And so if you just took this

kind of no-center style that this community settled on and put that in a 1,200-page book and that was something that young computer people could read that would be incredibly good for them.

Spicer: We're going to come back to the ARPA IPTO because the group and the trajectory of people that were involved in that group was, as you noted pretty well impressive, but I'd like to return for a second to why, especially in technical fields, in the past when it's acknowledged at all it's usually as a sidebar in the textbook, Ohm's Law or something. And part of that is of course necessary just because of the time constraints of getting a degree and so on, but—

Kay: Well, but at least Ohm's Law is in the textbook.

Spicer: Right, and I think—

Kay: We're talking about two different-- It's-- For instance, it's- it would be really great to have both Engelbart and his main ideas--which were not the mouse--The idea is that nobody takes the trouble to learn in a textbook, but if you're going to eliminate one of the other you should- you would eliminate Engelbart and at least put his ideas in there, the same thing. It doesn't matter whether it was Ohm although it's nice to know there was a guy with that name, but you don't want to force—

Spicer: Is there a distrust of the past among computer science and engineering people in the sense that—

Kay: I don't know. One time… One of the recent times I've tried the idea of even typing E-n-g-e-l-b-a-r-t into Google, which on the first page of hits gives you his web site with his 75 papers and the complete movies of the big demo 40 years ago, right there. In 2004, I could not find a single person who had ever typed those letters into Google. And so I was going around giving talks in computer science departments and so everybody had kind of an idea that Engelbart was the guy who invented the mouse but they had no idea why he invented it. And if they were physicists they'd be thrown out of the field because in real science you're not allowed to go back and just reinvent willy-nilly, and how many people are as smart as Newton was? And even a smart person doesn't do something smart every month so one of the ways of looking at it is really good ideas are incredibly rare, and the last thing you want to do is to try and get them back and you have a good chance of not getting them back as well if some genius in the past happened to be looking at it. So just the other day I was doing some interview and discovered this thing about Engelbart, and this guy was a user interface designer and when I suggested he go back and look at that stuff he says, "Well, isn't that cheating?" So he's confusing novelty with real creativity and that's totally pop culture. Pop culture could not care less about any good musician in the past because their identity is wrapped up on doing something novel, and they absolutely don't care whether it's any good because their thresholds are low.

Spicer: You have a very historical perspective in your writing—

Kay: I read a lot.

Spicer: Yeah, and I think it shows that you bring in references from renaissance scholars and writers. That's why—

Kay: That wasn't that unusual back then because almost everybody in that community back then had gotten at least one degree in something really established and really hard.

Spicer: Let's start with that then. I want to get to your personal life but we'll do that later, how you grew up and went to high school and so on, but let's focus on ARPA and ARPA IPTO and the role of that as kind of a clearinghouse for scientific and engineering talent over the last 40 years.

Kay: Well, it was everything. Taylor, who was very hard to get down from the mountain, was the third funder there of the four main guys in the sixties . It was Licklider, Ivan Sutherland when he was only 26, Taylor and then Larry Roberts, but at one… somehow they got Taylor up talking and somebody from the crowd asked, "Well, basically old farts always claim there was a golden age." And Taylor stopped and looked at the guy and thought for a minute and he says, "God damn it. It was a golden age." It was a golden age. Forget about how old we are. Just take a look at—

Spicer: What we did.

Kay: …what happened and try and understand why it happened, and it wasn't because we were any smarter than anybody else. That was… This is the classic example of unsophisticated perspectives on bricks, gives you piles of bricks or pyramids and it gives you walls and stacks and stuff like that, but you put the right architecture in there you start getting arches and you start getting gothic cathedrals and huge structures that are completely nonlinear in the synergy amongst the parts. And that's what Licklider actually set up, and it's very much due to his personality and so many people who liked him. He got that money because they liked him.

Spicer: Amazing what one person can do, and it reminds me of—

Kay: Yeah, and I didn't realize actually until reading Waldrop's book that it was as fragile as Waldrop because it.. when I was there as a graduate student and enjoying every second of it and a post doc and then going on to PARC it didn't seem that fragile. It was well established. It was at 15 or 16 places around the country. There was a thriving community and the community was doing things and everybody was happy as a clam and was <inaudible> basically and we knew that dimly… Well, I'd say I knew dimly that this was the greatest 'cause it was the greatest environment I'd ever been in, and- but looking back on it, it was cosmic because there's nothing like losing your first love, which happened—

Spicer: What was your first love?

Kay: Well, the first love was… in a way was the environment where you could do work, and so the funding went away in 1970 but coasted long enough for ARPA to do quite a bit with the internet stuff.

Spicer: That was the Mansfield…

Kay: The Mansfield Amendment, and then… but Taylor rescued some of the graduate students whose PhD's he'd funded and installed them at Xerox PARC so we got another certainly five or six great years and another few good years before the PARC situation deteriorated, but more or less from 1980 on it's been a different kind of deal and I'm convinced that it's more normal. There was a combination of this huge explosion of commercialization of these ideas but also the enlightened funders could not be found, and the people who gave the enlightened funders the enlightened money to do this disappeared.

Spicer: Right, and it reminds me there was a seventeenth-century Jesuit scholar called Father Mersenne that you might know—

Kay: Absolutely.

Spicer: --and he was a clearinghouse for mathematical and science knowledge in Europe for five decades or so, and he was the lynchpin and everyone sent their stuff to him and he distributed it. In a way, I think ARPA's role is similar to that.

Kay: Yeah. I wouldn't disagree but I think you would not be amiss by putting a grander cast because the cool thing about Licklider and the way he got the IPTO stuff started is that he had a number of realizations which Taylor, who was also a psychologist like Licklider was, Taylor went to school on and made them maybe even crisper. Xerox PARC was a little crisper version of ARPA than ARPA was, but… so Licklider had the wisdom… One of the things he said was, "We can't do our professional work inside the reality warp of the beltway so we want to have people from the community coming here to be deputy directors and hand out money but they shouldn't stay very long." So the idea—

Spicer: This was the idea that the people that should be handing out the money at ARPA should themselves have been investigators—

Kay: Yeah. So that was one idea, and the second one was that the ability to make great judgments deteriorated rapidly there and it wasn't good for people from the community to stay that long. So the idea is that nobody'll be, including Licklider-- So the idea is two years as the deputy director is it, and from now on we will spend six months to a year of our last year training our successor and then they'll do two years and then they'll do-- So that's—

Spicer: How were all these people brought together? Was it a shared vision? Did they go to the same schools? Did they have the same…

Kay: Well—

Spicer: …world vie…

Kay: Well, a simple way of looking at it is an enormous amount of good stuff happened because of the people who won World War II for us via technological innovation and particularly radar. So if you look at the… Radar was invented in other places in the world but for a whole variety of reasons it was made practical in dozens of different ways, and there's an excellent book about it at MIT, and the… Vannevar Bush having been a MIT professor, becoming President Roosevelt's science adviser, and he's well known for his role with the atomic bomb, but the atomic bomb was kind of a thing they could have demonstrated rather than actually using it but radar they actually had to use it. It was probably the critical technology in World War II, and I don't want to get too mistily romantic. But there was something about that generation of people who had come through the Depression, people who bonded together with the immigrants, some of them with enormous talents who came from Europe, the nature of the enemy, the simplest enemy we've ever had to understand, and the fact that as always the U.S. had enormous amounts of money and they were absolutely willing to spend it for a variety of-- Not the least amazing thing was that the whole atomic bomb project was just a little over two years. People don't realize that in the time. World War II we weren't in for any- near as long as we've been in Iraq already so… and Henry Kaiser was making literally a small destroyer or a small aircraft carrier a day when the stuff… So there's a great story that most people don't understand of what the U.S. actually brought basically to the level of engineering and production married with the more theoretical abilities that people think of as European and the theorists in our country studied in Europe because that's where it was going on. The combination of those two things was just incredible, and these people were different than my generation and really different from the generation of today. And I don't want to paint such a rosy picture of them, but I would call them wise. I would call a generation that could pull off the Marshall Plan just after we've lost so many lives.. Can you imagine… how did that…? So there's a great book about how the Marshall Plan happened and it was more fragile

than it looked from the outside, but how could we not have done that when the Russians were subdued? Talk about it now learning from history. The Marshall Plan was one of the great ideas of all time and it required a level of perspective that was able to see one of the worst enemies we've ever had as human beings who were going to do human things. And the idea is because people react to their environment we have to make a better environment and good things are going to happen from that, and boy, just even the last 25 years you can't find any politicians who really understand what that is and it's only money. That's what we do better than anything else. We're terrible at almost any other kind of thinking, but we can make money and we should just spread it around the world but that's another… So anyway, these people were special. That's my simple-minded thing. Like the people who did the Constitution who never agreed… But it was 55 peoplE… Only six weeks to do the Constitution. They didn't agree at the end but they got a Constitution that's worked for hundreds of years. They knew exactly how to work with each other with disagreements for this larger cause and so the general impulse of these people was wow, these people are enlightened.

So Licklider was one of these guys and so he very wisely realized that the worst thing he could possibly do is ever come up with a goal for the IPTO research, and if you look at when they put the “D” on ARPA part of it was to force goal-directed research and to save the goals and the proposals and everything else it was absolutely not the IPTO. Right. So Licklider's idea was, “I have a vision and the vision is that the destiny of computers are to be intellectual amplifiers for every person on earth, pervasively networked worldwide, period.” And then, because he wasn't a technical guy, which was another thing that saved this, he was a psychologist and so his idea was well, we'll let the technical guys figure out ways to do this and I'm going to fund everybody I think is really smart and I'm not going to care whether they agree with each other or not. So what he didn't have was a dogma and so if you went to the ARPA projects in the mid to late sixties, which was when I was there as a graduate student, you'd see every kind of thing. You'd see McCarthy's theory of what amplifying human intellectual endeavor would be and you'd see Engelbart's, and McCarthy and Engelbart not only didn't agree. They actually really didn't like each other that much and Minsky had an idea about it, and MIT itself had dozens of different ideas, and we had different ideas at the Utah ARPA project where it was very graphics intensive there because of Dave Evans and Lick didn't give a damn. And even better, Lick's successor, through a fluke that Ivan Sutherland got drafted, and they just absolutely did not want Ivan to go anywhere dangerous because most people who were even thinking about this stuff realized that the Sketchpad stuff was one in a million. It was just the most amazing… still the most amazing single PhD thesis ever done in our field and it's still amazing on absolute terms. It's just a fantastic—

Spicer: You mentioned that Licklider was not constrained or did not impose constraints—

Kay: Not really.

Spicer: --to whom he was giving money but how was he constrained? Did he have deliverables or—

Kay: Not really—

Spicer: --goals that he had to meet?

Kay: Basically, I think the most complete story, and it certainly jibes with what I knew from back then, was that the prototype of the PDP-1, which was kind of- sort of a commercial version of the TX-0 computer which was sort of a next generation version of Whirlwind--so it was all part of the SAGE effort,--the prototype went to BB&N. I don't know why but it went there. The TX-0 that was at Lincoln Labs went to MIT because they needed room 'cause Wes [Wesley Clark] was doing the TX-2 which was what Ivan eventually did Sketchpad on. And Licklider was consulting. He was at MIT I believe at the time and he was

one of the inventors of cognitive psychology so he, George Miller, Jerome Bruner and a couple of other people were trying to get psychology out of the white rat phase and start looking at the way humans think. So Licklider was a guy who everybody liked and he was also an influential and—

Spicer: Did that have any roots in computing, cognitive psychology—

Kay: Not then.

Spicer: --shaped—

Kay: It does now. That's what it means now actually, but back then, I think Bruner said something like that rats could do a lot of things but he had never met one that had written a line of Shakespeare, and so maybe not everything is extrapolatable from studying white rats. Maybe we should… And that was heresy back then but, like I said, these guys were larger than life. Jerry Bruner was so urbane, nobody… nothing bothered him. He just couldn't be more pleasant and smart and doing stuff and, like I say, the people like that got tremendously rewarded by the funders in those days 'cause the funders weren't looking for simple results. There was no Senator Proxmire back then and so they weren't looking for simple, countable results like the number of papers that were safe improvements on things or anything else. They were basically betting on people with ideas and they were kind of playing baseball so a .350 batting average on the 15 or 16 ARPA projects is a whole, given what they were trying to do, well, that was cosmic. So on the other hand, if you're uncomfortable with the batting average of .350 and you want a batting average of .950 or something you basically have to put adults and get them to play T-ball with three-year-olds and of course you'll bat a thousand, but what's the point? So the way the story goes is Licklider got very interested in the PDP-1 and started fooling around with it and because he was in another field, he had all this wonderful knowledge to make analogies to, and so the first thing he thought about was wow, this thing is not really like a hammer because it's interacting with me. And so he started thinking of it as something like a symbiotic organism that's helping us, and most non-biologists first don't know that we have a hundred trillion cells in our body but here's the astounding thing: Forty trillion of those cells, 40% of all the cells in our body, are bacteria, and you could think of them as parasites but they aren't. They're actually symbionts. Most of them help us digest our food.

Spicer: Commensal.

Kay: Forty trillion so even- and the largest cell count of any tissue that we have in our body is red blood cells and that's only 30 trillion. So Licklider… so his first little paper he wrote was called “Man-Computer Symbiosis” and he used this analogy, and it just struck him. He wasn't the first. McCarthy had already had… Basically, anybody who was enlightened, which there weren't that many but there were… So when McCarthy looked at SAGE, he could see what it could be for humanity and it was his first idea which happened somewhere between 1956 and '58 was wow… So his flash idea was kind of everything people labored to discover for the next 20 years, which was yeah, this is going to be the world's greatest thing and we can interact with it and we can do all of this stuff. And McCarthy immediately jumped to “what next?” which was we just are not going to be able to deal with the computer in the computer's terms. So he immediately came up with the idea of the “Advice Taker” that there needs to be the commerce between the human and machine has to be in some common sense terms. Now I cheated when I invented the PARC GUI. I wanted what McCarthy. I wanted an Advice Taker so badly and it might come up tonight because I needed something that would transcend putting adults in the loop for teaching children things. Adults are the biggest problem in improving education but I wasn't very good at AI and I had friends who were a lot better, and some of them were at PARC and so I thought we'll just let them invent this thing that can make a user interface that will be like a mentor, which is what McCarthy was talking about, but the common sense term things-- I said, "Okay. Let's-- So I'll make a tool base interface that makes a

theatrical presentation of computer things as objects that you can move around like you can in the real world." That's common sense also and it just doesn't-- it's reactive rather than truly interactive, and so… but McCarthy just jumped to yeah, here's what we need, and then less than a year later he had invented LISP in order to have the technology to build the Advice Taker. And like the Dynabook, the Advice Taker never got built.

Spicer: By the way, what is on page 13 of that manual Lisp 1.5 Programmers' Manual, MIT Press] that you pointed out?

Kay: Best thing you ever saw.

Spicer: Yeah. I saw the lecture you gave. What is tha?

Kay: So McCarthy was a mathematician. He got his PhD at Princeton and Minsky- I think he met Minsky there. I think Minsky was a little younger, maybe got his PhD just a little later than John, but John's- part of John's interest was in what Turing had been thinking about. And Turing's great realization is…and this is why they named the prize after him--is-- And it was interesting that his great realization came about 'cause he was trying to show a different route towards an already established result, but it just pointed out that it took hardly anything to get any kind of machine that was at all interesting to simulate any other kind of machine that was at all interesting. And if you think about that in terms of mathematics, and Turing did and McCarthy did and Church did and Emil Post did, what that gives you is a meta-mathematics because you've got a mathematical technique now for dealing with some mathematical apparatus including the symbols and the inference rules and all of that stuff. And so McCarthy wanted to do something that would allow you to do symbolic computing rather than just numerical computing, and I don't think it was really a complete afterthought, but as a mathematician you couldn't help wanting to show that this formalism that he had come up with for dealing with symbols could be a universal Turing machine. And there's a lot of different ways of doing that and thank God John picked the hard way, which was to try and realize his own symbol system in itself rather than just pick some already known universal Turing machine and to show how you could do that. So the result that it gave you two things. One is it gave for the first time in a really… Part of the idea of making up languages in mathematics is to make the result small enough to see with just one eyeful so Maxwell's Equations was much bigger in Maxwell's original way of doing it, but the Europeans [soecifically, Oliver Heaviside] quickly found that if you came up with three operators, divergence, curl and gradient, and vectorized those equations they would collapse down to the famous four on the T-shirt. And when you look at those four on the T-shirt everybody noticed that the electric field one and the magnetic field one were not- didn't look the same, but in fact in Maxwell's theories they were supposed to be the same and that was really puzzling. And the puzzlement that that gave rise to more or less immediately produced the special theory of relativity from Einstein because that was… He said the fact that these things look different is some sort of artifact of us not looking at things the right way. So what McCarthy had done just as a kind of a trick was to get half of one page to sort of say, "Okay. Given that we have this little symbolic language now, what if they write the program in it to execute its own programs and what would that actually look like?" And it came out really small, and in fact it can be a lot smaller, and…

Spicer: So there's a beautiful self-similarity and recursion that's-—

Kay: Well, and it's one of these things where-- Yeah. I hate-- You don't want to divide the world into two groups, but a small number of people who got fascinated with this for one reason or another--and I got fascinated in it from a really weird--The last thing I saw was the bottom of page 13 'cause I-- Minsky had written a truly wonderful book called "Computation: Finite and Infinite Machines," and in there he was explaining the original Gödel way of dealing with this problem that you can generate more mathematical

facts, more relationships faster than you can prove them. And there's a lot of different ways of doing that but Gödel used powers of numbers as a data structure because he-- what he wanted to do was to be able to encode mathematical formulas into numbers and then prove something about the numbers and that would also prove something about the formulas. And Minsky did this in a particularly cool way in this book of instead of doing it the way Gödel did it, he made a LISP [program] in which the memory for the LISP structures were actually numbers raised to powers. And that alone is just awfully cool.

Spicer: Insight. Yeah.

Kay: And so I was going through-- And I was a mathematician so I was going through that, loving every second of it, and at some point Marvin mentions in the book, "Well, this is what LISP is." And I thought well, wow, that's just neat, and then… so I went from there to reading… And I knew about LISP but I didn't know what it was and so then I found this thing, and then people either got fascinated with the small amount of code on the bottom of page 13 or they don't. If you got fascinated with it, for me, and I've talked to many other people, it's basically a rainy Sunday afternoon and when nobody's around to bother you where you use every finger you have because this thing is highly recursive so you've got your fingers down in all of the places there where you're trying to get this thing into your head—

Spicer: You're keeping the parentheses-- Is that what you're doing with your hands?

Kay: Yeah, and it takes—

Spicer: You're mating the parentheses.

Kay: It took about two hours and then there's this-- Now I want to- I don't want to say sexual but there is something and it's a long, drawn-out "aah" kind of thing where you see that oh, yeah, this is… and this is why people do math because if you don't get that… Math is too hard unless you can participate in the joy of the relationships just work out so perfectly.

Kay: So basically-- so you could tell stories like this in using the tropes of Zen, you know, when such and such happened and the student was enlightened.

Spicer: Right.

Kay: So besides all the joy from that, all of a sudden you've got a way of thinking about really powerful computation because the big achievement maybe was not a trick and that is [that] nobody had ever done a universal Turing machine with that much slope. So, one of the things about Turing machines is they don't do that much. You know, they're usually weak, little--but very complicate--little things and universal Turing machines usually don't do that much. And what McCarthy was able to do was to get one where the lovely trick happened to be more powerful than ALGOL, because this was more powerful than the most powerful language of its day. And so you got the most powerful language of the day plus the fact that you could now think about the most powerful language of the day in terms of this, which meant you could immediately start thinking of more powerful meta-things to do in a way that simply wasn't possible when looking at 50 or 100,000 lines of machine code for a programming language, and like I say, some people found that hugely instructive. And for me, it came along at exactly the right time because I'd already had the object idea and had been doing things with it. But the meta-nature of LISP was not… I had about three-quarters of it, but I wasn't as smart as John. And when I saw LISP I realized, "Oh yeah, this is… you should just really never think of a programming language and this is one of the absolutely non-dispensable ways of thinking about what a programming language is." And I wasn't that interested in functional computing but the whole technique that McCarthy used was fantastic for doing the object stuff. And so it

allowed a lot of thinking about what objects were going to be and allowed us to do our first version of Smalltalk at Xerox PARC in just a month. So, and similar things happen-- you know, one of my great heroes who works at the computer museum off and on is Steve Russell and he was the guy who look-- and you should get Steve to do one of these things because--

Spicer: He did LISP on the PDP-1… no that was Peter Deutsch…

Kay: No, no, he did--

Spicer: No, no, that was Peter Deutsch

Kay: Peter Deutch's thing was really significant. I'm going to mention it tonight. But Steve Russell did the first LISP.

Spicer: On the 704, I think, or 70--

Kay: I don't-- I think they had gone beyond the 7. I think it was probably a 709 by that point, but McCarthy proposed LISP, I think, originally for the 704. But I think they… we can ask Steve. He's going to be here tonight. But basically, you know, as McCarthy tells the story, and it's a great story the way McCarthy tells it, that Steve Russell looked at this eval [a LISP keyword] and he said something like, "Oh," he said, "you know, that's a program. If I were to program it, then we'd have one." And McCarthy said, "And he did, and we did." And I had a very similar thing happen with Dan Ingals who looked at--

Spicer: He said the same thing didn't he?

Kay: He said-- more or less, yeah. Dan's line was, "You just do it and it's done."

Spicer: Yeah, I like that. Even though most of the people at PARC, for example, had Ph.D.'s, you still had a pretty practical bent, I think, wouldn't you say?

Kay: Well, I think the… yeah, it's a good… so I'd characterize actually a lot of people in the ARPA community, and particularly the people at PARC, as first, generally speaking, being able to do both hardware and software. Generally speaking, haven't gotten a degree of one kind or another, maybe not a Ph.D., but of one kind or another in something difficult that was established. And the reason that's a good idea when you're in a new field is it's hard to tell the bullshit, you know, what's the quality level supposed to be. And the nice thing about getting a degree in pure math or physics, like Butler Lampson did, or electrical engineering, like Chuck [Thacker], is these were all tough fields. You know, there isn't any air guitar for any of these fields and so the-- it was very helpful for everybody to have something to use to criticize efforts in computing. And then I think the other thing was that everybody was either an engineer or had engineering as a hobby of some kind. You know, it just went along and I think people who got fascinated with the computers back then were already fascinated with cause and effect relationships, sometimes mechanical or electrical or biological or some… they were fascinated by…

Spicer: By systems.

Kay: The idea… yeah, systems, things made out of parts that interacted and you… like, with Arches, [?] you got something more than a simple sum of the parts, and we saw some incredible examples of that. Like Butler, who is simply one of the most amazing people any of us have ever met, when-- and I won't go through the whole story, but it's a great story, that we tried to buy a PDP-10 in the first few months of PARC and Xerox was selling Sigma… what used to be Sigma… yeah, Sigma. They used to SDS

computers but Xerox had bought them and so they didn't want DEC out there taking pictures of these black PDP-10 boxes going into the Xerox then using them in ads because they were already contending. So Xerox said no and so PARC almost broke up at that point.

Spicer: Really?

Kay: Yeah, because our thought was, and there was probably only 12, 15 people at that point. But the thought was, "Well, it isn't up to them. We need a PDP-10 because that's what the ARPA… is on the ARPA net and we're part of that community and blah, blah." And so in these meetings that were held, it turned out that the reason Xerox had bought SDS at all was because of Butler and Project Genie at Berkeley.

Spicer: Oh yes, timesharing.

Kay: Yeah, and so SDS was not responsible for the SDS-940. It was actually done at Berkeley and Taylor forced SDS against their will to make 940's. So, the first round of timesharing computers in the ARPA community were those things. So, this is an irony. And because of that, Xerox, without really looking too clearly into where all this technology came from, bought SDS.

Spicer: Right, at quite a premium too.

Kay: Yeah, and then there it was when we showed up at PARC. And so we had one of the best operating systems designers in the whole world there and so part of the meetings were something like, well, it takes about three years to do an operating system and I don't think Butler wanted to do another one right then anyway. But then the idea came up: but we could build a PDP-10 in less than a year. I don't know who uttered that thought first.

Spicer: Yeah, it seems so implausible almost.

Kay: Everybody started… well, not really because we had Thacker and the BCC-500, which was the project that maybe 10 of these guys had come from, had done a micro-coded timeshared computer. So the idea is, you know, we'll emulate a PDP-10. We won't make… and by the way, we'll test out the first integrated circuit memories. We won't use cores, and that led to a whole bunch of interesting interactions with Intel about the 1103 [memory] chips. But to get to the point of this story, and there are many wonderful stories about this machine, but the cool one was that Chuck was going to do the memory and the memory architecture and the console and everything else and he just really didn't think he had time to do the CPU. And Butler said, "Well, if I don't have to use an oscilloscope, I'll design the CPU." And Butler never had done anything like this before. But, of course, he knew…

Spicer: Because it's too much hardware for him?

Kay: No, he didn't want to do anything analog.

Spicer: Oh, I see.

Kay: He was a very confident guy and with good reason.

Spicer: That it would work from paper?

Kay: Yeah, because Butler was very confident. And so his idea was, you know, what's basically sort of what's the big deal, it's just a bunch of logic gates. And so Thacker set up the design rules for Butler so

he never had to use an oscilloscope. But it was complicated. It was 14-phase clock. And by God, Butler designed the CPU for this thing and it worked.

Spicer: Wow, yeah, he's in a league of [his own]…

Kay: That was one of the most amazing things I've ever seen because the learning curve he… and the whole MAXC was done within that first year. And the Alto was done so quickly, just three months, by Chuck and just two technicians, in part because they already had gone through the whole learning curve of this 1103 memory and had these modules. And so the Alto was a triumph of parsimonious design beyond anybody's wildest dreams, but it had almost nothing in it. So, this is sort of Thacker at his absolute best and…

Spicer: Is it 160 ICs or something?

Kay: 160 MSI ICs, two cards, 80 on each.

Spicer: And that was the whole Alto.

Kay: That was the whole… yeah, in fact…

Spicer: Wow, that's amazing.

Kay: Yeah, and I was a little disappointed on the thing some years ago. There was a thing on the Alto that Butler and Chuck presented and the Alto is just an absolute gem of a design and it's one of those things that somebody could write a book that would enlighten people to just… although Chuck is doing something quite as beautiful in this BEE-3 thing. You know what that is, the BEE-3 [Berkeley Emulation Engine], which is essentially a computer. It's a piece of hardware to do the modern equivalent of micro code, which is field programmable gate arrays. And he got permission from Microsoft to get a Silicon Valley company to just make it and anybody can buy it. It's more or less out now and this thing is just, you know, it's like an Alto. It's got something like 70 gigabytes of memory and absolute shitloads of FPGA's in there in this beautiful… so Chuck is doing it because he has some things he wants to try out. And as he points out, nowadays, it's actually harder to do a computer in many ways than it was because you really have to do a lot of integrated circuit work just to try an idea out. Because the Alto is made out of MSI, if you take a genius, like Thacker, and you could go down to Fry's and because Fry's in those days sold components, and you could just, you know, get a bushel basket full of stuff and the genius could make a computer in a few months. So, Thacker's idea is 'we need to get people back into the joys of computer architecture' and the way to do it is to make them a thing that just has enough--and it's not that big physically, but it's just absolutely loaded with what you need to be able to program up truly. See, I mean, you could program a supercomputer on this thing that's highly parallel and very efficient and, you know, we're going to be using it heavily in the research that we're doing. And it's done in conjunction also with Dave Patterson's RAMP project at Berkeley and--

Spicer: So it's called the BEE-3?

Kay: Yeah, it's called BEE-3, the Berkeley something something, because there was a BEE-2. And this thing got kind of hatched between Chuck and Dave Patterson and I sat in on a couple of lunches of the thing because, you know, the future of programming languages partly has to do with whether you can execute them in any reasonable fashion or whether you're going to spend all of your time optimizing on terrible hardware. And so the Alto was this incredible thing where… we did think about a few of the things that it was going to have to do, but we didn't think about them in particular because it was all micro coded.

But a lot of thinking was put into what it was generally and there are ideas from the TX-2 because again, Ivan--the TX-2 had 22 program counters and zero overhead tasking between them and the interrupt system was set up to connect directly to the selective sequences on this machine. So, there was no graphics display on the TX-2 there was just a point plotting oscilloscope. And so the first thing Ivan did was to actually program the display equipment he wanted for inventing Sketchpad. And in fact half the resources of the TX-2 were used just because it used one 36-bit word for every point that was plotted on the screen, so half the memory was used for just doing that. But, you know, if you needed to change his display you'd just go in and change the code, and the same thing was done on the Alto. The Alto had 16 program counters and every time the Alto started getting expensive, Chuck would throw away some hardware and speed up the micro code processor so you could get those functions back as code. It was a truly elegant--

Spicer: I want to start on the Dynabook a bit and but just pick up, you mentioned that I think in one of the articles that you find the current state, or even the state of microprocessor architectures is pretty parlous, I think, in your judgment. Is that fair to say that?

Kay: Well, I don't think… there was some effort… well, I think everybody is entitled to their point of view, right?

Spicer: Sure, yeah, but do you have any specific--

Kay: I just don't think making money is enough of an end goal. But if you talk to Intel or Motorola, they'll say it is enough of an end goal.

Spicer: Right, and your main critique is that they didn't, at least, involve enough software people where people architected their computer in devising these chips.

Kay: You know, to say it may be too bluntly, so bluntly that it might not even be completely true, Intel and Motorola just were not interested in software at all. And I can prove that's not true myself because Intel, in the early '80's, did try to do a higher level processor called the [iAPX]432 and they should have stuck with it because everybody who tries doing a higher-level architecture seems to make the same mistakes on the first one, so it's like the omelet you do to clean out the pan. And what they should have done instead of saying, "Well, it can't be done," which is false because it had been done several times earlier, and to say not to get discouraged about it but to actually think about it further and get somebody like Thacker who can, you know, the difference between the FLEX Machine that I did with Ed Cheadle and the Alto that Chuck did was just, you know, 98% of the good ideas I had in the FLEX Machine were just gone. Because the problem is, you know, when you have a chance to do hardware you tend to over optimize and that's what happened with the '432. And Chuck's idea was you only want to optimize in a few places and then optimize the shit out of it and really make it possible to get those… and that was a huge revelation to me. So, but generally speaking, you could say that Intel saw themselves as a device… like, they didn't even make package memory for a long time. They never made a computer, right, so they didn't see themselves in the larger business. They were making a lot of money just putting more transistors on chips. And we were using their components and stuff and we begged them to take a look at these microcoded architectures. And pretty soon, though, it was too late because once the IBM PC with the Microsoft software got locked in as a standard, as bad as it was, it didn't matter because Intel started making oodles and oodles of money and you could have this kind of erosion, you know, the erosion model of learning, which is once the little rut gets made kind of randomly, the water tends to go there whether that's a good idea or not. And in the business sense, they were making money hand over fist so it was impossible to talk to anybody that had money as an end goal and tell them why they were wrong. So, but yeah, so the--

Spicer: One thing you pointed out once was how a definition of responsiveness, which is intuitive to a musician, for example, playing a flute and you blow in the flute and it immediately produced a sound, but we still have computers that don't do that in spite of gigahertz clock speeds.

Kay: Yeah, my favorite thing is to show… in most talks and I'll probably not be able to resist tonight but it's really great to watch the movies from the old Engelbart demo. You know, the display didn't look that great but they got sub-second response. And when I give a talk to a general audience I always ask them-- well, on that slide, which has some movies embedded, I have a picture of Engelbart and his group having a meeting and that picture happens to require the total amount of memory that was on the SDS-940 that Engelbart uses, only 192K. So, it's got a half MIP machine and 192K and it's a timesharing system, he's getting sub-second response and I always ask the audience of computer kids, "How could they do this? Why?" And only once have I ever gotten the right answer and the answer was they would rather have died than not get it, basically. One kid, who was a sophomore at UCLA a couple of years ago when I gave that talk, you know, he basically said, "Because they wanted to." Because most people think there's some law of nature that presents and they can't do the arithmetic that if they're on, like, a one or two gigahertz machine, even if it's incredibly inefficient with Intelisms everywhere, it doesn't-- you know, it's more a matter of what your standards are.

Spicer: Is that a reflection of some of the system level goals coming not necessarily from engineering or computer science but from cognitive psychology or human factors?

Kay: Well, I think the desire for this stuff comes from, you know, trying to solve problems at some human level. But I think that the psychological problems with getting people to try and solve them for whatever reasons are deeply built into our genes. So, anthropologists have this idea of human universals that there's never been a culture that has been found without a culture, never been a culture found without a language or without stories or without interests and power or kinship relationships and so the anthropologists have-- there's about 300 things that seem to be in human genes. But there are plenty of cultures that never have reading and writing, never have an idea of equal rights, so these things we can conclude are more like inventions where the other ones are kind of more built in, and one of the things that definitely-- one of those universals is the idea of coping. So, if you're.. because human beings have had to cope pretty much until the last few hundred years, unless you happened to be a king. And so, you know, re-normalizing to any situation no matter how bad and then getting by is built into us and it was necessary for our survival. And the idea of progress had to be explained in the 17th century.

Spicer: Right, because change was coming.

Kay: Yeah, and when the constitution is being thought about by Jefferson and Madison and George Mason and other people, one of the ideas that came up, which was a new idea then was, "Hey, the world is going to be different even 20, 50 years from now, so maybe we should put into the constitution that there should be a new constitutional convention every 30 or 50 years." Fortunately, they didn't do that, but that was radical because what they realized was going on in the industrial revolution was happening that people were not going to die in the same world in which they were born, and that was never true in human history before. So, I think that what happens is unless you're crazy and this group was… could be thought of as people who were driven by aesthetics more strongly than other… as Butler says in his talk on this stuff, he thought the most interesting thing was that nobody else was working in personal computing at the time… nobody, not even in ARPA. And even our colleagues thought we were crazy. And one couple of ripples out from the ARPA community they thought we were completely nuts. But the aesthetic, so it was one of those things, like, there's a First-Order Theory in computing, "You should never do your own tools," like, you should never do your own operating systems, you should never do your own programming language or your own hardware, and the First-Order Theory is true. But what's interesting is that the

Second-Order Theory is exactly the opposite and is also true, "If you can do your own tools, if you can do your own hardware, if you can do your own software, if you can do your own programming languages, then the world is different." So, both of these theories are true, right, one of them is more risky than the other, but the risky one pays off in allowing you to invent a whole new set of things because you're willing to take care of the entire infrastructure yourself and that's what the two dozen people… but people can't believe that most of the big inventions at PARC were done when the group was only about 25 professionals. It's just… you know, because there was a lot of software involved and we had this wonderful dictum that Butler had given rise to and more or less forced everybody to sign in blood. We fought him for a long time but he had a real will of steel and the idea was that, himself included, that he just really didn't feel that we should do demos as a result. So, his idea was, "Let us agree to never do anything that isn't engineered for 100 users." And so this was controversial and--

Spicer: Isn't that in conflict with the personal computer idea in a way?

Kay: No.

Spicer: Oh, because they were networked….

Kay: Yeah, the idea is that if it's a net-- if we invent a network, it has to run 100. If we do a timesharing system, it has to run 100 users. If we do a personal computer, we have to make 100. If we do a piece of software, it has to be documented for 100 users. You know, so it's just apply the 100… and basically, you cannot do a bubblegum and baling wire demo and scale it out to 100. And so we fought him like mad for a couple of weeks and gave in and this turned out to be like magic.

Spicer: This concept.

Kay: Oh, man. So, it subtracted about a year and a half from every project because it forced everybody-- because nobody wanted Butler climbing down your throat. So, it forced everybody to think about engineering things that you never think about when you're doing prototypes because Butler was going to come by and beat you around the head and shoulders and so the result of the thing was that everything was engineered just a lot better. And it turned out that everything got done quicker because a lot of the problems we normally didn't solve ahead of time because we didn't think we needed to, we actually needed to. So, a lot of those things that you don't do because you think you're going to get to the demo sooner actually gets you to the demo later and vice versa. So, that was awfully cool.

Spicer: And less debugging, too, you would think because of more reliable hardware.

Kay: Yeah, everything-- so, that was magical. And then, of course, there was this idea that because there weren't a lot of people there and Xerox was kind of helping inadvertently that way because for a long time personnel was allocated by slots and so each slot was worth the same whether it was for a secretary or a technician or a Ph.D., and there weren't a lot of them. And so PARC sort of over-- you know, we used most of them for what would be called principal scientists. So, we didn't have a lot of bottle washers or anything around, and but all of us had been… you know, I had been… I worked my way through school as a programmer and so most people liked some of the nuts and bolts stuff. And so the idea is can things be organized so we can just do this so we don't have to hire a whole bunch of staff. And this is always tricky, but basically the lack of resources plus the backgrounds of a lot of the people really helped because the… and particularly for our stuff, the fact that we had this world-class hardware designer and everybody agreed that something like a higher-level architecture was absolutely what you wanted to have in order to get higher-level languages and our stuff was premised on we were going to do a language that was as simple as LOGO for children that because of its object-oriented nature and a couple of other little ideas in

it, you could also write the operating system. And so it was going to be a very, very simple thing that could touch most parts of computing. And the first version was actually called SLOGO , "Simulation LOGO," but it also had this nice idea that it was going to be pretty darn slow. On the other hand, Thacker was going to build a machine that was going to give you maybe 50 times the response of a timesharing terminal. So, all of a sudden you were in this interesting place where there is a sweet spot where you could do something it would not take a lot of code to write and it would be really inefficient, but on the other hand the hardware was meeting it. So, instead of spending all of your time optimizing this awful hardware, you just wrote microcode created as close to the ideal environment for this language to run. And microcode was set up so that it didn't care whether the environment was for Smalltalk or LISP or for the Mesa or any of this stuff. They all had somewhat similar run time architectures but they weren't the same, but the differences weren't locked in in the hardware. So, everybody did their own what were called 'virtual machines' back then and every time Thacker did a new machine type there you sat down and wrote microcode for it and the body of stuff that went along with these languages just traveled from one machine to another and it would sniff the machine that it got to and pick the microcode for that machine and download it and it would customize the machine to this environment.

Spicer: That's very interesting.

Kay: Could be done today, amazing idea.

Spicer: So, Dynabook, I guess we better start talking about that.

Spicer: So let's chat about Dynabook and where the idea came from and what shaped that. And are we there yet? I guess that's another question.

Kay: The irony, of course, is it's the world's simplest ideas… I'll try and convince people tonight. And it's had an interesting history. Chuck Thacker likes to call it the most influential computer that has never been built. But basically the root for it was pretty simple. So the personal computing idea, I think, was most easily seen in what Engelbart was doing. He was on a timesharing system, but he was showing us personal computing more or less as we know it today and in some cases better. So they could do collaboration kind of a bit, and they had some stronger theories about what it meant to work in groups and so forth. People will eventually re-invent this stuff. And the romance of personal computing was in Sketchpad. That was like the most romantic of all systems. And a second… why romantic?

Spicer: Yes.

Kay: Because well, Engelbart was essentially showing us commerce between people in a variety of different ways and group knowledge and other stuff like that. And Sketchpad was something that could be used for something like that, but it was essentially showing us what computer art was, which is the art of making simulations of interesting ideas and understanding those interesting ideas better. And the particular way Sketchpad went about it was spellbinding, because it at once had this idea of 'you should be able to just draw on your notebook' your ideas. And the computer should be able to understand something about what you're drawing. And you could advise the computer by saying here's the relationships between this… This thing isn't aligned; it's really… it's part of this crankshaft thing I'm building, and so this thing is connected to that thing. And this guy is anchored at a point, and this guy… this is supposed to be equidistant from that, and so it has these degrees of freedom if I hook this other guy to it. And so you just do the things that you would be thinking about as the relationships between this thing. And Sketchpad would be able to sit down and to solve the implications of those relationships, because you're not programming it sequentially. You're not programming it in terms of algorithms. You were programming it in terms of the desired relationships as a series of constraints that had to be

satisfied. And one of the very clever, wonderful things that Ivan did was if you under-constrained something, like making a crankshaft, Sketchpad would show you all the solutions. And so it would animate--

Spicer: As you moved the light pen, it would animate?

Kay: No. I mean, it would… if you under constrained the thing…

Spicer: Oh, I see.

Kay: It would just show you all the solutions, so an under-constrained crankshaft, Sketchpad didn't know which position it was. And Ivan, of course, once you-- I think this probably happened accidentally the first time. But then he realized it would be really cool if he made it come up with each solution near the last one, instead of giving you random solutions to the crankshaft thing. And it turned out that the solvers that he used in Sketchpad were at their very best when they were looking for nearby solutions to something. So you look at Sketchpad and you think, man, that is… so Sketchpad is another thing that hasn't… it's hard to find something today that has gone beyond it to the next useful thing, because there's so many things you can do with computers by not being as idealistic and creative and inventive as Ivan was. So again, it's this coping thing. There's a million things you can do, particularly if you're from a culture in which a member of the culture that these ideas were being put into is a culture in which most people who had any power at all did not do their own typing even. So there's kind of an indentured servitude idea in a somewhat class-oriented society about who did what.

Spicer: This came up later when PCs came out and executives refused to have them on their desks.

Kay: Exactly. And so yeah, then they became a status symbol, but they still don't really use them. So these things appealed much more, as they do today, to people who are used to doing their own thinking. And this includes artists, but it also includes mathematicians and scientists and engineers. And so most of the good stuff that's using computers for what they're absolutely indispensable for is being done in technical fields or in art. And basically the business world is using them for very boring things like automating databases from the '50s. They aren't really that different. They don't understand that simulation could be a strategic weapon for them and so forth. And then the next most romantic thing to Sketchpad was a system called Grail. And most people know that Engelbart had something to do with the mouse. But the really superior technology, the technology that was a miracle, happened exactly the same year as the mouse, which was the Rand Tablet; 1964, Tom Ellis… when I went to graduate school at Utah, Rand Tablets cost like $18,000, which is like $100,000 today. If you had a Rand Tablet, you had arrived in graphics research; it was that simple. They were all handmade, and they were just incredibly good. It's just one of these things where what the Rand Tablet could do was just miraculous. And these guys had big ideas, and those ideas were built on the first great end-user system, which was called JOSS [Johnniac Open Shop System] that had been done by Cliff Shaw. And JOSS was the first time anybody with real taste and real determination had decided to make a real system for end users. And as I'll point out tonight [at CHM public lecture], one of his lines in his paper he wrote in 1964 was, "making an end-user system is just the little things, the hundreds and hundreds of little things." And he was the first person who did those hundreds of things.

Spicer: This is Cliff Shaw?

Kay: Cliff Shaw, and hardly anybody has been willing to do those hundreds of things in almost any system since then. So that system, when you sat down to it, it was like you were a cat and somebody was

stroking you. And just the knowledge that you were going to be able to use it for an hour in the afternoon, because there weren't that many JOSS terminals that Rand was-- like, oh my God.

Spicer: And it was a timesharing system or not?

Kay: Yeah. Originally, it was eight…[user stations]… an important part of the thing was they actually modified IBM Executive typewriters. Actually, the first JOSS system was nicer than the second one. But in both cases, they made the typewriters. So the first one, they took an IBM Executive typewriter, and they had gotten this idea that since the people who used this had notebooks that the continuous paper that would go through this thing would be already drilled. So it was drilled with three holes. And then because it was an IBM Executive typewriter, you could have two-color ribbons. So they got the idea that it would really look nice if that you typed in one color and JOSS replied in another color, and that would make it just so, so nice. And yeah, it was eight terminals. I never got to use that system. But then they wrote the best documentation that's ever been written, just fantastic, every kind of documentation of every size for every kind of user. So you were in--

Spicer: There's a real clarity of formulation when one person… you talk about that with your agglutination versus <inaudible> the programming styles.

Kay: But a single person can have bad taste, but in this case you've got this guy who said about two words a year, who is not a big talker, Cliff Shaw. I revered him, because when they were going to decommission the Johnniac, he asked that they give it to him. So they actually kept it going, and hardly any computer has probably been worse to program than the Johnniac, in every possible way, if you've ever looked at the instruction set on it. It was slow. But basically, it didn't matter, because this guy had a real sense. And he didn't try and make it do more than it could do. Everything it could do it did perfectly. It did well. It did things like on this tiny little thing he had a little pattern matchers to look to see if certain kinds of relationships were being used and whether there were rounding errors or not. And he would fix those, so it was just every little deal in there. And so the use of the thing back then, you felt exactly as though somebody was taking care of you. There was a light that showed you whether the central system was working. No other system had… every other system, years later. It was just, is it slow? Has it crashed? But in JOSS, you knew, knew exactly what was going on. And again, it was like McCarthy's LISP. This appealed to some people greatly, like me. And other people didn't care about it, because for a lot of programmers, part of the joy is their being able to surmount something that's ugly. And I never got one iota of pleasure in my entire computing career from surmounting something that was ugly. But the people who like that have no interest whatsoever in making a system better, because the worse the system is the better they feel, because they are the ones who can make it do things.

This is the difference between a hacker and non-hacker. Hackers delight in finding a way around some horrible feature. And non-hackers don't want to have horrible features. It's just two different worlds, and this is why Nicholas Negroponte was such an incredible user interface designer in the '70s is he was trained as an architect. If you're an architect, you don't make a house full of exposed nails pointing out on it, unless you're a person who delights in being able to avoid every one of those nails when you go running to the bathroom in the middle of the night. Everybody else wants a house that doesn't have those nails sticking out. And so you could divide… the theories of what this stuff is, there are people who… the predominant theory of user interface in the '60s was it was access to function. And the examples that everybody had been coping with, particularly since there was a lot of military funding stuff is if you go into a nuclear submarine, you've got something that requires 20 people to run it. You can imagine it being easy to make a nuclear submarine where one person could run it. But they never did, because that wasn't the way you ran a submarine or a power station or a nuclear reactor control panel. So all of these things are functions; you have to get trained to use them. And I had gone through that when I was in the Air Force.

The Air Force was all about getting people to be the interstitial mechanisms that made functional things usable via training. And there's some musical instruments that are like that too, where they just weren't designed for anything. Some of them are accidental even. But then the other way of looking at it is, and I don't know who said it first; it could have been me, or it could have been Negroponte or somebody. But the other way of looking at it, if Licklider's dream is real, then we're creating something we're going to live with. It's like a house. It's like clothing; it's going to be part of us for hours every day.

Spicer: This is the opposite of the previous vision of access to function.

Kay: Yeah, so number one thing there is can anybody learn how to use it? And how many people can learn how to use it? So immediately… and to me, this is the one… because we hardly invented anything new in user interface at Xerox PARC. But the outlook was completely different, and partly because of the Dynabook and partly because of Seymour Papert's influence on the Dynabook, that children for the first time in our way of thinking were full-fledged users of this system. In fact, they were the most important users, because adults are terrible at learning new paradigms.

Spicer: Is that the main reason that you worked with kids versus adults?

Kay: Oh yeah.

Spicer: When you were testing?

Kay: Well, there's a whole bunch of reasons. One is I was very… when I was seven, I accidentally found out about the Holocaust in a really graphic way by finding a *Life* magazine and… it had actually been published in '45, but it was hanging around the house. I found it in '47, and it had pictures of Buchenwald in it. And I was a very sheltered kid, and we didn't have television to get kids used to violence or anything like that. It shattered me. And when I could think about it again, one of my thoughts was oh, that's what fairy tales are about. I had this dim realization that the ogres, the giants and the ogres in fairy tales were actually adults. And in fact, that's what they are. There have been some studies about what are these things mythologizing. And they're mythologizing--

Spicer: **"**The Uses of Enchantment?"

Kay: --what adults do to children.

Spicer: Bruno Bettelheim.

Kay: Bruno Bettelheim, exactly. And so I got quite frightened of adults and actually didn't get less frightened as I got older. And of course, school was terrible, except for a few exceptions, because I'd read too many books before I even went to first grade. And school is not about multiple points of view, and if you have multiple points of view going in, you're in deep trouble. So I really didn't think about education until I'd been doing this desktop computer called the Flex Machine with Ed Cheadle at Utah and presented it at the first ARPA Graduate Student Conference in 1968 at Allerton House at the University of Illinois. And part of that first graduate student conference was a tour of the University of Illinois. And there was the PLATO project and Don Bitzer's first couple of little plasma panels. They were about this big. And the thing is, the flat screen display screens had been in science fiction for a long time. They were in *The Shape of Things to Come*, the movie, and they were talked about a lot in the '50s, particularly in popular science, that in the future we'll have displays that hang on the wall. So this was not a new idea, but nobody had ever seen one before, so there was some discussion. And we knew how many transistors were in this Flex Machine; there weren't a lot. And so a thing that wasn't quite the Dynabook got talked

about that summer, was "yeah, someday we should be able to put the transistors in a thing like the Flex Machine on the back of one of these displays. And we'll have a little computer like this." And that was exciting, but it wasn't above threshold. Someday it will happen. Then a few months previously I'd heard Minsky give one of his diatribes about education. It was great. And he kept on saying Seymour Papert's name in it, because they were running the AI project together at that time. And so I determined that year I was going to go see what Papert was doing, and of course I read the papers that they'd written. And so about 40 years ago, maybe last month, I visited Papert's school setup that he had. And the thing that got me there was the level of understanding these 12-year-olds actually had of the LOGO programs that they had done. I was very, very lucky, because I got there before the Turtle had taken hold. And it's not that the Turtle is a bad idea; it's a great idea. But the problem is it leads to a lot of air guitar, because the teachers can be fooled by the children just driving the Turtle around with simple program--

Spicer: When you say "air guitar," you mean just an illusion?

Kay: Yeah, the form and not the content. And like whether my hair is combed or not. And this kind of did LOGO in, in the '80s, because the early adopters would take it in and pretty much everybody was happy if the children were using it. And they really didn't pay enough attention to whether the--

Spicer: Which is pretty low bar.

Kay: Very low bar, and of course, Seymour knew exactly what they should be using, every different part in there. But the thing I saw was a little bit of Turtle, but the Turtle there was the very first mechanical Turtle. And while this was being installed and everything, the kids were using LOGO to do things like writing little programs that would take any English word and make a Pig Latin word out of it and then take any English sentence and make a Pig Latin sentence out of it. And some of the kids were writing fairly decent little translators from French into English and vice versa, because they were taking French as their foreign language. And the nice thing about those programs is in the LOGO of the day they were quite elegant. They were little recursive programs. They were kind of like classical LISP kinds of programs. And when I saw that children really had gotten fluent in this stuff in just a couple of months, that was when sort of my previous set of theories about personal computing and stuff just shattered. And I spent the whole day there, and I spent a lot of time talking to Seymour [Papert] and Cynthia Solomon, because what I was seeing was that equivalent of what the printing press brought to manuscript writing, which was a different way of expressing ideas. And when I bring this up, people always say well how can you say that, because you can write in a manuscript anything that is printed. So how can you say the printing press brought anything different? And the answer is that one of the keys to the printing press was that once an author corrected the galleys, the printing press would guarantee to make thousands of exact copies. And this was not true in manuscript culture. And so the actual rhetoric used in manuscript culture remains story-like and even used allegories and other kinds of things that would survive small errors. So it remained kind of a transcription of oral culture except in a few important cases. Whereas, as Erasmus realized, but not very many people in the early 16th century [realized], that this was a whole different ball game. And Erasmus was one of the inventors for page numbers for books, which had not been in any book in Occidental culture until 65 years after the invention of the printing press, because they didn't use page numbers for sequencing pages and didn't for another 100 years. They sequenced pages using--

Spicer: You just trusted that--

Kay: For a long, long time, coming from the manuscript culture and then into printing even into the 18th century-- I have a bunch of 18th century books that are sequences--ways that at the bottom of the page, the first couple of words of the next page were put there.

Spicer: Oh yes, I've seen that.

Kay: So you could go through, and page numbers and those devices were coextensive for 100 years or more before they finally got rid of them. So Erasmus and Aldus put in page numbers, apparently so you could refer to things from within the manuscript, not refer to things in other books because typical editions were only a couple of thousand copies. So you couldn't be sure the pagination, but you could be sure of the pagination in your book. So these things were used to do the equivalent of hyperlinking arguments and refer back reliably to what you had said before. And furthermore, you didn't have to use the same kind of rhetoric. You could start thinking more along Euclid's "Geometry" terms. And so that one of the biggest changes in expositional writing happened all the way into the 17th century and where a whole bunch of writers--but Thomas Hobbes was one of the early ones--decided that the old story way of writing was not the way to do what you really wanted to use devices that were more like Euclid and less like stories. Like you didn't want to really try to justify a point by referring to authority, which was a favorite thing before. Instead, what you wanted to do was to get the reader to agree with some premises and then just use logic to lay everything else out. And this was a whole new style of writing, and it was coextensive. Bacon used this style. He was one of the early guys who used this style. And it was coextensive with the invention of science and the Royal Society. And you can still see the old style, because Galileo wrote in the old style. He wrote in terms of characters talking to each other that is rather like what Plato did rather than what Euclid did. So the printing press could actually handle a level of argument that no other writing system could ever handle before, and it could spread it around. And we all knew that the important thing about a computer was it had a different way to argue than anything ever had before, which was by simulation. So instead of making claims, you actually put the model there. And it's not the same as putting a mathematical equation, because you're actually giving the person that you're making this argument to the actual mechanism for seeing what the argument actually is and trying different cases but also modifying it and seeing if they can falsify it. And so that had already been thought up, and I and other people had thought of those ideas from reading McLuhan, because McLuhan basically demands that you try and ask that question about every medium of communication you have.

Spicer: And about your tools and how they shape yourself.

Kay: Again, that didn't seem that… that was what computers were. But when I saw children being able to get fairly seriously into this level of stuff, that's when I went crazy, because there wasn't any pathway between what a great… what a computer could do better than anything else and an ordinary human being. But here it was; this is what Seymour… so for me, that was the big "holy shit," and it was simple one that just wow, this thing is really the next 500 year thing after the printing press. And we have a glimpse here of what it means for children to be able to get into the really powerful stuff of the last 400 years, decades ahead of when they would be able to do it using regular stuff, because the other part of this stuff--and this is where the huge message of the Turtle--The Turtle is actually a vector, and it's programmed using vector ideas but put in terms of the child's own body. And so that's what's called differential geometry, which was invented by Gauss in the 19th century. And it's basically the differential geometry of vectors is basically the language that science used. And Seymour had shown how you can relate this completely to the way the children think about their movement in the world. And thus, you should be able to teach them… like one of the mathematics they should be able to learn relatively easy, more easily than something like analytic geometry or even ordinary arithmetic should be this differential geometry of vectors. And that's proven to be the case. In the world of calculus, the computer does that thing that calculus had to go to algebra for, which is you've got it downstairs in Babbage's engine. It can perform the integrations of simple additions directly. You don't have to dick around with abstractions and complex relationships. You can just sit down and look at the differential relationship of something in terms of simple addition, which is one of the few things having to do with arithmetic and math that's actually built into our minds. And without going beyond simple addition and subtraction, you can get as many orders of differential equations as you need. And

most of Newtonian physics only uses two, those first order ones that give you increases and decreases at a steady rate. And then the second order ones give you all the quadratics, which is sufficient to do virtually every Newtonian relationship there. And it's so much simpler. And again, everybody knew this, but Seymour… this is why it's so sad that he's not here tonight, really sad, because he would be the guy we would be honoring. But you know he had a terrible accident, that's why he's not-- oh.

Spicer: No, I didn't know that.

Kay: Three years ago, he got run down by a motorcycle in Hanoi. He was at a conference in Hanoi, and apparently the street traffic is …, and he had part of his brain removed to keep him alive. And he hasn't recovered.

Spicer: That's a shame.

Kay: Yeah, so yes, so he would be the guy here, because he had these fundamental insights. So basically on the plane ride home, I drew that little cartoon. And when I got to a cardboard model… this is the one I made for Japanese television about 15 years ago, because the one I made at Utah I gave to the chief scientist of Xerox, and he never gave it back. But this looks pretty much like it, removable memory and the stylus. And the cover was not put on it until PARC, and it didn't get the name Dynabook until PARC. But I was very curious as to what a good shape was. I made a variety of different shapes and came up with this notion that you're always carrying more than one thing. And so portability means being able to carry something else. So there are a whole bunch of shapes where you'd only be able to carry that, but making one of the dimensions thin seemed like a good idea, and I was curious about the weight. So the original one had a hole that you could pour lead pellets from a shotgun into, and the answer there was about two pounds was the most anybody really wanted so schlep around. It's still true. And [I] calculated how many pixels it should have on the screen by thinking about basically counting characters in various books and stuff and trying to understand. So this is the world's simplest idea in the sense that it wasn't seeable, because there had to be some catalytic reaction to tip things over the edge. But Moore's law was 1965, and we had the Rand Tablet, and we had Sketchpad, and we had our desktop Flex Machine. We had flat panel displays being worked as… when I saw the Bitzer plasma panel, I looked into it and found out that RCA was doing an LCD panel. And Peter Brody at Westinghouse was starting to do a panel. So it looked like that technology was going to happen. But to me, the thing that drove it home was it was unthinkable to have the extension of Seymour's ideas not go to children. And it was unthinkable not to make a highly portable device for children, because children are mobile. You don't want to tie them down at a desk with a desktop computer. That's why I drew that cartoon. The kids are outdoors doing stuff with it. And so it was the focus of the kids, and the reason the kids meant so much to me goes back to my trepidations about adults, that I just didn't think adults were very good learners, and I didn't think of them as very enlightened, et cetera, et cetera. But children give you a chance, every generation, to bring them up in a richer environment, and maybe they'll turn into richer people and more enlightened and stuff like that. So from my standpoint, except for the fact that I had to write my thesis about this desktop machine, that was the last time I ever thought about that desktop machine, because it just wasn't in this special space of making this special kind of learning medium for children. And the reason we don't have a Dynabook today is not from the hardware. The hardware, although I don't think a lot of the hardware designs are particularly nice; it's trivial to do every aspect of the hardware basically 100 times better than I had thought would be necessary back then. But to me, the Dynabook was always a service idea. And besides the media services, which were completely obvious, the main service was to be an environment for learning things on. And there are basically four parts of that. There was what is a powerful idea? How can you teach a powerful idea, with or without computers? What does it mean to have a mentor be a human being to help children? And can you get around the lack of good mentors by making a user interface that can mentor? So those are kind of the four-- and then the idea was this stuff

is as important as reading and writing. So it doesn't count if only 10 percent of the kids get good at it. So we basically have to go beyond whether kids are interested in this stuff intrinsically or not. We have to do something like reading and writing where regardless of the talent that the child has we have to find pedagogies that will help at least 90 percent of the children in the population get fluent. So that led to 25 or more years of complete failures with little successes here and there. So it was only about 10 years ago that we came up with a mixture of stuff that suddenly started working with a wide variety of children. And with the right kind of adults in the loop, it was fulfilling basically three out of the four criteria.

Spicer: What kind of advice would you have for young people?

Kay: Marvin Minsky says "You don't understand something unless you understand it more than one way." So part of what the modern outlook on knowledge is that you have to have multiple perspectives in order to view things and the scientific take on knowledge is that science is a process for in part for dealing with what's wrong with our brains, we're always misinterpreting things and seeing things through our beliefs and so forth and to some extent we have to but to the extent in which we can get around the really bad errors that have been made for the last 100,000 years is astounding. So the last 400 years is a testimony to what various kinds of technique and thinking, real thinking skills as opposed to fake thinking skills reveals a completely different world than the one humans had thought they were living in. And so one way of thinking about the advice is that science is not the same as technology, science is a different way of trying to deal with how things can be known and the ways we should think about that knowledge and it's really a shame that science is taught as the physical sciences rather than as kind of the center of the education system. It's the probably the greatest idea that human beings have ever had and so another way of looking at it is if you had an IQ of 500 but were born in 10,000BC you're not gonna get too far except maybe you could outwit everybody. But Leonardo was smarter than most people have ever been and he couldn't invent a single engine for any of his vehicles because he was born in the wrong century. What that says is that knowledge almost always trumps IQ. Henry Ford was not as smart as Leonardo but he was born in the right century and so he could make a bunch of vehicles and the thing that trumps knowledge is outlook. So people knew a lot in the 19th century but what they knew as voluminous as it was, was not exactly obsoleted but was put in its proper perspective in a stroke by Einstein because he was able to take a different outlook and the world was revealed to be something much stranger even than the 19th century.

Spicer: Yeah and even he had trouble <inaudible>.

Kay: Another Minskyism is that you can't teach a cat calculus, so it's possible that you can't teach human beings the universe. Right, we might not be smart enough to think up the extra brains which are what, another way of thinking of what powerful ideas are, the extra brains that help us think about these things that cave man couldn't think about. So it's not clear whether-- but as far as advice to young people, it's hard to do most kinds of problem solving in a logical thinking and creativity unless you have something to map between. So just sitting down and learning one thing is probably the biggest, quickest dead end especially if it's weak.

Spicer: This is a critique I think you have of people in technology, sometimes they are victims of their own success, they're very good at the very narrow field but they can't really flip out <inaudible>.

Kay: Yeah I think you know if you take stem and rearrange it in historical order you get technology first then engineering because you can get technology just by pure tinkering and engineering is an attempt to make things I think using more principles and then mathematics was invented by the Greeks. Then finally we got science just a few hundred years ago and these days you're at your peril if you try and do any of these without mixing the other ones in. So I think another way of characterizing the ARPA and Parc

people is that they're pretty good at all four of these and they were pretty good at knowing when they were doing one and not doing another, you can put on one kind of hat and not another. So it's a pretty well known thing that once you start optimizing something it's very hard to have any design thoughts because you're just in a completely different world and you're gonna be satisfied with different kinds of results than when you're trying to design or create something.

Spicer: Who are your role models?

Kay: I think my biggest role model as far as a person would be Dave Evans, just he was a guy that you can only thank by trying to treat others the way he did. There's these special people are-- you can't thank them because they're not-- what they do is not done for any kind of thanks, so he was a very special guy and then the other part of that question is really complicated for anybody who reads a lot. So and I appreciate even the terrible books I've read because you have to read a lot of books in order to get a few thousand that are worth reading and in a way it's hard to appreciate what those are without looking at a lot of bad ones.

Spicer: And the same with software I guess.

Kay: Yeah although in ordinary engineering out in the physical world when you build a suspension bridge, you've got the very nice property that the stuff you're building the bridge from has mass and it's on a planet that has gravity and the combination of those two and maybe some forces like those winds blowing down the Tacoma Narrows for the famous bridge collapse there. But basically engineers are human beings, scientists are human beings and the difficulty of actually building complex things or finding out complex things in nature is partly policed by nature. So science has this nice thing, it doesn't matter what a scientist thinks because the only ultimate critic is nature itself, you have to submit it back and doesn't matter whether an engineer believes that structure thus and so is a good structure and if nature pulls it down it's a not a good. And the problem in computing is that the materials of computing don't have any mass to speak of and the computer doesn't exert any gravity to speak of and so you can make truly awful things that are rather, rather large. That they don't come crashing down but I think most software that is around today is in the midst of like a ten year crash. It's crashing because people can't take care of it, it's getting bigger and it's getting slower, it's getting worse and everything, but it's not crashing in a dramatic way and it's not endangering most people's lives and so the people who do this are quite happy except for the ones that react aesthetically to it or really don't wanna be slaves to it, really wanna do something better, they'd like to modify it in some good way or replace it in some. But in fact an ecology has been set up in which there are millions of jobs where probably millions of people are not needed but in fact the-- and it's kind of like one of the worst things we have had on the planet for a long time is coal burning power generating plants. There's a whole industry built around them or a really bad modern thing is ethanol from corn. It's just really a bad idea, there isn't any good part of it and but there are plenty of people who support it, it has huge political clout, there's zillions of people who don't understand that it's bad. Sounds good until you understand what it actually is and stuff like that and if ethanol goes away in the next five years I'll be the most surprised person the planet, even though it should. Because especially being an old molecular biologist the right way to-- like the first thing you don't wanna do is make alcohol because the energy density on it is not very good and secondly it happens to be soluble in water and so it costs you a lot to get that alcohol out of the water. What you really wanna do is make Alcan's which are like basically what most petroleum derivatives are, they're essentially hydrocarbons that they're insoluble with water and they happen to float on top of it and so if you make a bio digester that will take something like glucose or cellulose and produce Alcan's and that's what they're doing. While you just have something that's a lot like gasoline in the top of these columns the bottom is the fermentation is going on and you just stick a tap in there and you can pretty much put it into your gas tank. So it's gonna take a long time for the political pressure and the awareness of politicians even the ones who wanted to do the right thing to actually

understand what that is because they're a part of the largest-- 95% of Americans really can't hold a conversation with a scientist. They're just not scientifically literate enough to do it and so it's very difficult to get them to understand the three or four things that you need to understand to pick one energy strategy over another.

Spicer: What was your proudest moment in your life?

Kay: Geez I don't know, I never think of it that Freud has a theory about (which I don't necessarily subscribe to) but it's kind of eerie that people who are artists that the impulse to art grows out of a child's fascination with their feces because it's one of the first thing a child notices that they create and they play with their feces and they draw with them flush them down the toilet and stuff and so the-- so I say I've been most proud of being able to hang in there when an idea is good and worthwhile being worked on. So I had to learn how to do that and it's not a natural talent I have, I like to start things but I learned from my ARPA community and particularly from Parc where we had a beautiful mixture of people who could all play basketball essentially and some of them were better at leading the charge down the floor and some of them were better at taking the shots and so the basic deal in computing I think is it's a kind of mathematics but it's a finite mathematics that is enormous and classical math is usually about infinite things but tiny and so the number of degrees of freedom and the size of the things that are like implications and computing means that we don't get to prove anything particularly interesting about it even showing that something is NP hard is not that interesting because most of the things that we're trying to do with computers we find ways to do it, it's a mixture of things. But on the other hand computing is kind of BS if you can't make it run and so kind of the parallel to what proof is math is taking what seems to be a fruitful idea in computing or a theory in computing and making a working system that exhibits this thing. So you're kind of debugging a little more than proving and so to me in computing there's a huge engineering component and proving something in math has some engineering type components into it but I think computing also has this thing that except for really unusual circumstances the nature of what it means to create something interesting in computing, almost always involves more than one person so it has this-- and part of that's the engineering nature of it and it's not so you need a community but you also need working groups and the dynamics of these working groups are basically and the dynamics and the context of the community are the things that make these things work and so in a way we mislead people when we give awards to individuals even though I wouldn't take away a single award that's even been given to Butler and so I think that the deal here is that there are these special people, there's no question but every single one of 'em at least from the ARPA community will say without any collusion beforehand will say "Well it was the community that made us better than we were and better than we would be in an ordinary group without this context."

Spicer: You triggered another question which is did you feel any Cold War, were you working with-- in the sense of defending the country?

Kay: No, I mean.

Spicer: There's no panic like after Sputnik at your level anyway?

Kay: Not it's the nature of that funding that it was throwaway money, it was money leftover form the space race when NASA-- so basically after Sputnik, the job of matching up to the Russians was initially given to DoD and ARPA was the funding agency on that and somewhere around project Gemini which is the two pilot where NASA had been set up and that whole thing got translated to NASA and there was money left over and it was that money that was-- or some of that money was given to Licklider and according to all of the stories, almost too good to be true but done in the Cosmos Club where a lot of

those deals were done in Washington and very likely with cigars and not a woman in sight I'll bet. A bunch of old boys sitting around the table Jack Rowena and other, Bob Sproull senior I think was at that table.

Spicer: What else might you have done?

Kay: Oh I think lots of things, I kept on saying I was gonna back into molecular biology because it's an interesting parallel field that kind of blossomed the same time computing did and it has a lot in common with computing in many ways and I've more or less kept up on it although certainly the last 15 years it's just been-- I can still talk with molecular biologists in most areas but I definitely am not up on it. Now and my particular interest in there has exploded the last 15 years, which was morphogenesis which is how you-- because what you studied when I was studying molecular biology in the early '60s was basically cell biology and one, even bacterial biology but development of-- for anybody who's interested in systems, how an embryo develops is one of the most interesting kinds of things and an enormous amount has been discovered just in the last 15 years about it. So I think that's going pretty well there. But I think that the way-- for instance in computing there are a lot of ideas that came easily to me and one of the areas I wish I'd had even one good idea in was AI and I don't know whether it's my biological background or what but aside from the fact that object oriented stuff helped AI along never had a really good idea in AI and I regret that more now that mentoring interface that we need to complete the Dynabook idea and I felt it so deeply when Nicholas Negroponte did the OLPC thing and my heart sunk, you know, I was thrilled that he was doing it because if the mentoring situation is bad in the US, it is horrible in the third world and so basically you have this thing where you're doing a couple of good things, you've got something that is cheaper than paper, you've got something that is likely to resist the horrible climates it's gonna go into better than paper. But it doesn't even have a program on it that can teach a person to read their native language without requiring a teacher. It's like the simplest thing.

Spicer: So you're always gonna need that team?

Kay: Like if you're gonna do one thing on a machine that's gonna go into the third world, it should be the idea that whatever it can contain, just the elemental thing of being able to find out what it's user knows and be able to teach them to read so to me that's the flip side of all the romance of this stuff is a lot of these hard problems that were identified and that one was identified way back before I even got into computing, they haven't been working on. Basically the low hanging fruit has been more attractive and you can make a lot of money on the low hanging fruit and the funders have been very adverse to funding these things that might take 20 or 30 years. It's not unlike the space program so that what got funded there was a gesture, get to the moon in less than a decade. But space travel was omitted and it probably set space travel back 25 or 30 or more.

Spicer: Yeah it seems to have.

Kay: Well we're still using technologies that are somewhat similar to what were used for the moon shot and you can't get space travel out of them, so everybody knows that. And I think some of the same things like it's very difficult to get people to even imagine the things you could get them to imagine about what computers could do 40 years ago. Because we have computers today and computers like television make it hard to think about anything else except what they do, do. It was much easier when there weren't computers around, they were just mythical giant brains in rooms that you could get people to imagine "Oh a computer could do this and here's why and a computer could do this and here's why." Nowadays people are convinced that computers can't do a whole bunch of things because they don't.

Spicer: Yeah I think there is a maturity that's coming from just a couple of decades of average people using computers.

Kay: Yeah people get normalized and most people are unsophisticated enough to think that they're normal is reality and that's the first goal in education is to remove that bad theory from student's minds. We're set up to normalize because our nervous system likes differences and so we disappear the things that are constant and this is what McClewan was saying and so the most interesting thing about any person are all those things they're not aware of that they think are real and aren't.

Spicer: So we're all in a collective sense in a refractory state waiting to wake up?

Kay: Yeah and that's the purpose of art.

Spicer: From perpetual stimulus.

Kay: The purpose of art is to remind us that our normal isn't reality.

<crew talk>

Kay: So I guess an analogy there is if you go into a science museum, the chance that you're gonna see any science is almost zero because what you see is technology and so you will not see like Newton's principia is the real deal but you don't see it on the wall of any science museum and I actually got asked to write a little essay for the San Jose whatever it was Museum of Science and they had an exhibit of I guess 75 paintings that had science themes and so my little 800 word thing was titled "It already is art" and it was all about that science museums have failed at presenting science itself and as the art that it is because they-- and so the computer museum is fun here but I think it's way too biased towards old 19 inch racks and what's interesting about computers is that you can program and there's hardly anything here or the version of this in Boston that tries to engage people in what that means in any of 19 different dimensions you could choose. For instance the bottom of page 13 of the LISP manual, if you can't find that in a computer history museum, what is that museum for, it's one of the greatest things ever done. If you can't find an explanation of not just what Sketchpad looked like but what it is and was and what it means, what's the point or Inglebart, so I think this museum started out as a solution to the problem of Gordon Bell's garage because that is how the Boston version of this thing happened. I love this old stuff, I love to walk through and see all this old hardware that I used to program that's fun but I think from the standpoint of calling it the Computer History Museum, it's not called the Computer Hardware History Museum, it's called the Computer History Museum then I think it's missing the essence of what computing is about and so when you have something wonderful like the recreation of the PDP1 and Space War on it, Space War can be written in just a couple of lines of code in a modern programming language that could look very much like English and so you could actually side by side you could actually show that program and you could let people make changes in the program. So there's a million things that could be done to even put the historical stuff in context and in a form in which somebody in a museum could use it. Another thing that I judge museums by the quality of their shop, so the shop here is not good, should be able to get all kinds of things as takeaways. You could imagine giving everybody an RFID card or even a PDA when they come in here, so they now have a user interface for something and that could give them a presence when they go home, get on the website and be able to explore. So these are both comments about the way most museums can port themselves, they have this-- because I've consulted with a bunch of museums and the official term is release time, which is usually no person should be allowed to look at any exhibit for more than two minutes because there are other people coming along and two minutes is the wrong amount of time to learn about anything. You can only learn about news, you can't learn new right and so I think the job of any museum is to start off with news which is stuff that is incremental to what people already know, that's why they came here and find a way of getting them to get involved with the amount of time and the depth that it takes for them to understand something new. So that's my comment, so I love that this museum is here, I followed it every step of the way because Gordon is a good friend of mine and I was

thrilled when he got this building for a song during the last financial crisis but and the quality of what is attempted here is high but I don't think enough of the right stuff is attempted here, it's a low hanging fruit thing.

Spicer: Maybe we can chat later.

Kay: I think it's a real challenge because so at other museums I've consulted for I've pointed out that and this is a little different because this is a smaller scale but the time you really get to learn something in a museum like the Metropolitan or the Chicago Museum of Science and Industry is when you go to the cafeteria. So what you want is like the Met has or had until they got rid of their fabulous-- they had their main cafeteria right next to the book store. Go to an exhibit, get interested in something, go to the bookstore, get something cool, go to the cafeteria and next few hours you spend perusing this and if things are going well you don't leave the building, you go back into the bookstore, you go back in the museum and right so it's like how to do a good library and the answer is well figure out some sort of cafeteria, do what borders did. This place is maybe a little small for that.

Spicer: Actually we're gonna have a café.

Kay: Yeah well if you are then most of the thought about content should be put into that café because that's the only place where people actually have time to think about any of this stuff. So when we were at Disney we designed tables that either had top projectors or bottom projectors that made the table top into an information source and we used the PDAs as the user interface mice and stuff and so you could imagine you get to the café, this thing knows where you've been.

Spicer: Did you hand out the PDAs?

Kay: Yeah, yeah, you can fix it so they won't-- we did this in the parks also, you can fix it so they can't steal them, they don't work anywhere except in the museum. You get them when you come in and you hand them out when you go away. But now you've got everybody with the.

Spicer: That's interesting, with a common interface.

Kay: Yeah, yeah and this I've been touting this idea for ten years because when I look at an interface, because of my biases my first question is "What can I learn, how can I learn this thing, am I learning a shape, am I looking at a color, is there something functional here, is there something I can do to test out a theory I have." Most museums fail this horribly.

END OF INTERVIEW

