Oral History of Butler Lampson

Interviewed by:
Alan Kay

Recorded: August 22, 2006
Cambridge, Mass.

CHM Reference number: X3697.2007

Alan Kay: Part of my job here as given by the Computer History Museum is to try and get a few good words from you that we could use as the opening blurb for your award from the Computer History Museum. But also to get an oral history.

Butler Lampson: I was going to say, I thought the job was to record hours of brilliant conversation that historians in 2100 will pore over.

Kay: That is your job. My job is to only to try and instigate it. My theory about this thing is that you should not try and talk short.

Lampson: Well, we've got lots of time right?

Kay: Okay. We do have lots of time and tape is cheap.

Lampson: Tape is cheap. Right. My sister's a film editor and she hates it. She says things were much better in the days when film was expensive, because people would think about what they shot. Now, she says, they shoot hundreds of hours of crap and then they expect the editor to sort it out.

Kay: We have to transcribe those hundreds of hours.

Lampson: Yeah. Somebody's got to look at it, it's got to be fussed around with, and besides, she says, frequently in the whole of hundreds of hours you don't find what you want because nobody thought about it beforehand.

Kay: You remember Bonnie, my wife, ran a film and video company for ten years.

Lampson: Yeah. So she knows all about this. Right.

Kay: So my job is not to say much, which I'm going to work on. Of course I have a lot of the questions that the Computer History Museum would like to ask you.

Lampson: Okay. Well shoot.

Kay: And I have a few special ones that I've always wanted to ask. But I think a good way to start here is: I and many other people regard you as one of the most significant characters in the romance of computing over the last 40 or so years, and we now have finally have a chance to talk to you about it. Some of the questions they have are ones that are sequential, and some are ones that will branch out. The simple sequential one is: I think you were born in 1943.

Lampson: That's right.

Kay: Where were you born?

Lampson: Washington.

Kay: Your father and mom: what were their names?

Lampson: Oh. You're asking these formal questions.

Kay: Yes. Yes.

Lampson: Their names were Edward and Mary.

Kay: What did your dad do?

Lampson: He was—well, I'm not sure what he was doing in 1943. He was somewhere in the Army. But after that he was a Foreign Service officer, so I traveled around the world a bit when I was a kid.

Kay: So you were pretty much one of the last kids to grow up in the age of radio rather than television. Do you remember any earlier radio shows?

Lampson: Well, when I was growing up… When I was three we moved to Turkey and that was in 1946-47. Turkey was pretty primitive in those days. If there was radio, I don't remember anything about it. The earliest thing I can remember that had anything to do with radio was that my parents were listening to the 1952 democratic convention nominate Stevenson.

Kay: Yeah. I remember that.

Lampson: Which I wasn't personally interested in at all, of course, at the age of 9.

Kay: But you must have read a lot as a child.

Lampson: And television-- well, it was when we went home on home leave there were these television boxes. That was really magical. But then escalators were magical, too.

Kay: Yeah. After Turkey you went back to the U.S.?

Lampson: No, actually we went from Turkey to Germany, and we lived in Christian[?] Dusseldorf, where I learned to hate vegetables, because Dusseldorf was part of the British zone and I was sent for a year to a British army school where they had all of the honorable traditions of the British Army cooking. I've recovered from most of that, but I still can't eat brussel sprouts.

Kay: I can't either. Of course the joke about New England is when you call up your mom and say, "Hey mom I'm coming home for Thanksgiving." And she says, "Great. I'm going to put on the vegetables right now."

Lampson: I've never heard that connection with New England. Oh dear.

Kay: I grew up here, which is very, very, very similar to the British system of making sure they're nice and gray.

Lampson: Well there are variations on that. My son and his wife were traveling in Nova Scotia. They stopped and bought some fish, and then they went to some kind of campground and there was a grill but it wasn't in very good shape. They thought they would grill the fish. They went and talked to the management and asked, "Could we get some wood or charcoal or something and fire this up." And the lady said, "You know, it'd be an awful lot of trouble to fire that grill up and put the fat on and deep fry the fish. Why don't you just do it on the stove?" The idea that you would grill the fish was just not in her mental framework at all. But anyway. After that we lived in Bonn, which is a part of the American zone. That was the headquarters of the occupation. The Americans had decided that it was not healthy for American dependents to associate too closely with Germans. So they built a little replica of a midwestern town. Well, it wasn't exactly midwestern because it was actually a low-rise apartment buildings. But it had a shopping center, and a school, and a church, and a little club, a movie theater. Everything you needed. There was really no reason ever to go outside of it. So it was a little bit weird.

Kay: It must have had a library.

Lampson: It had some sort of a library, yes. I don't have any really clear memories of that.

Kay: One of the questions I've been asking people and I've always wanted to ask you is: did you have any childhood fictional heroes?

Lampson: I don't know about that. I used to read a lot of the children's classics that were very much out of fashion in the fifties, like the Oz books for example. I don't if you know this but it used to be considered unprofessional for libraries to stock the Oz books.

Kay: No. I didn't know that.

Lampson: Uh-huh. So if you looked for them in public library you normally you wouldn't find them. My parents used to summer in Maine, in Castine, and for some reason the Castine library escaped this rule and they had a complete set of Oz books which I used to devour when we were up there.

Kay: Any heroes from those books?

Lampson: I don't know about heroes, no.

Kay: You could be kind of like a real Wizard of Oz, but maybe that's stretching it.

Lampson: Well, the Wizard of Oz was a pretty mixed character.

Kay: Maybe like some other people [we know]; maybe Bob Taylor? Somewhere along the line you got interested in mathematics and science.

Lampson: Yeah. That must go back a long way, because it goes back as far as I can remember.

Kay: Books about that?

Lampson: I guess, yeah. I mean there were kids biographies of scientists and things like that that I can remember reading.

Kay: By 1960 you would have been 17 or so.

Lampson: 16. Right. I graduated from high school in 1960.

Kay: Yeah. Had you had any contact with computers?

Lampson: Yes, because when I was in high school it was quite close to Princeton, and a friend of mine had found a [IBM] 650 in a small lab in Princeton and somehow managed to talk the lady who ran it into letting us use it because it wasn't being used very intensively. So that's when I started to program. It must have been in 1958.

Kay: The 650 was a drum machine.

Lampson: It was indeed.

Kay: And it ran faster if you calculated where the instruction should go when the…

Lampson: That's right. There was a program that did that for you, fortunately. A few people would do it by hand, but for the most part you relied on this program.

Kay: I never programmed that machine, but didn't it…

Lampson: It was a great machine. It was bi-quinary internally, which meant that a decimal digit was represented by a bit that said whether it was less than 5 or not, and then 5 bits. I never understood the reason for that. But I read something a few years ago that explained that the reason they chose its representation was because of its error detecting properties.

Kay: Exactly.

Lampson: If any bit flipped, it would be illegal.

Kay: And you know the [Grafacon] tablet at Rand was bi-quinary for exactly that reason: because it was noisy. So it gave them a little bit of extra margin on the thing.

Lampson: I guess things were expensive in those days, because the console display was bi-quinary also.

Kay: I've often wondered, because from what I've read about the 650 there were other machines that had drums and had addresses on most if not every instruction. The Alto at [Xerox] PARC also had something…

Lampson: Well it was very standard to do that for microcode, because it was faster than running an adder. And also you could play this game of doing branching by OR-ing things into the low-order bits of the next instruction address. So it was for totally different reasons.

Kay: So you had gotten somewhat infected by computers by the time you went to university.

Lampson: That's right.

Kay: You went to Harvard and you made majored in Physics.

Lampson: That's right. Well, Harvard was not exactly a center of computing. It had been earlier, with Howard Aiken. But by 1960 the university's central computing facility was a UNIVAC I. And it was really great. I remember the first time I saw it, in the old Aiken lab, which is a pretty up-to-date building. It had an enormous lobby about, I don't know, three or four times the size of this room. You walked up the stairs. There were big doors. You walked into the lobby, and it was all dimly lit. Down at the far end of the lobby there was a receptionist desk which was lit up, but the rest of it was all dimly lit. This big room, basically empty. On the right-hand side there was a glass wall. On the far side of the glass wall was a machine room, which was much bigger than the lobby. It was also dimly lit. All down one side of the machine room was the Mark I relay calculator, which was commissioned…

Kay: I've seen the picture. Yeah.

Lampson: …I guess [in] 1940. There were 6-foot cabinet after 6-foot cabinet of relays, and readers for the two-foot wide paper tape that carried the programs, and giant cables connecting the things up, and so forth. All turned off, of course. All down the left side of the room was the Mark 4 relay calculator, which looked basically the same. Then all the way down at the end of the room, where you could see it through the mist, was the University's central computing facility, which is this UNIVAC I. A whole bunch of tape drives that used, believe it or not -- do you know about this? -- the UNIVAC used metal tape.

Kay: Yeah. Some of the early computers even used wire, because people did sound recording on wire.

Lampson: Yeah. That's right.

Kay: And they used wire for…

Lampson: I never experienced that. But this was metal mag tape. I never programmed…

Kay: Did you use one that had mercury delay loops as…

Lampson: Yeah, I did. Right.

Kay: So this is a ten-year-old computer by then?

Lampson: Yes. I told you that Harvard was not in the forefront of computing by 1960. I never programmed that machine. But it stayed there more or less in that state until the fall of 1962. Then the workman came in and they ripped out all of Mark 1s and Mark 4s and put raised flooring in the machine room and moved in a [IBM] 7090. Then the university was modern. But that was a big revolution. I never programmed that machine either. I used to do my programming at MIT first.

Kay: Let's see. Tony Ottinger was at Harvard fairly early?

Lampson: Yeah. That's right. He was there. And Gerry Salton was there.

Kay: And Tom Cheatham?

Lampson: He was later. He was after my time.

Kay: So you got your degree in Physics.

Lampson: Although I actually I did quite a bit of computing because I found a PDP-1 that belonged to the Cambridge Electron Accelerator. I got hooked up with the Physics professor who was trying to write programs that would interpret spark chamber photographs, and I did a lot of programming for him.

Kay: How much memory did that machine have?

Lampson: The basic PDP-1 had 4K of 18-bit words. This one I believe had 8K. It might have even had 16K. It was a fairly heavy-duty PDP-1. It had tape drives and other fancy things, and connections to experimental equipment.

Kay: [The PDP-1] was the fourth or fifth computer I learned, and it was the first one I thought was small. I thought it was impoverished until I got to like it.

Lampson: Well, compared to the 650 it was pretty expansive. I had programmed the 709 and 7090 in the meantime, but that was a much more remote experience. I learned to program in Fortran from Peter Wegner, who at that time was a graduate student in economics, I believe, at Harvard. Yes. He taught this informal course in Fortran programming.

Kay: Somehow I always thought he'd come over a little bit earlier…

Lampson: Well, as I say, he was still a student registered in Economics at the time, so I don't know to what extent he was actually deeply involved in computing. But he did teach this course. We would ship our stuff down to MIT to get it run. On one occasion there was a giant foul-up and we were actually allowed to put our card decks into the online card reader of the 709 at MIT, and that was a great experience. That was when I learned that the online card reader only reads 72 columns. And I hate it. And the reason for that…

Kay: Was it the sequence numbers?

Lampson: Well, no.

Kay: Because they used to those.

Lampson: They did. But that wasn't the reason. The reason was that the 709 and 7090 were 36-bit machines, and the way the card equipment worked was that when you read a card you got 2 times 12 36-bit words, which were all the bits of the first 72 columns. But it would have been a big hassle to do the other eight columns, so they just didn't bother with that. Unfortunately my program, although it had sequence numbers in the last eight columns for the code, used all 80 columns of the data. So it didn't work at all. It was quite baffling for a couple of hours.

Kay: I remember the first time I used an [IBM] 1130 and tried to get a character off the keyboard, it was the strangest looking thing in the register, until I finally realized that the character encoding on the 1130 was the BCD card code.

Lampson: So you got 12 bits?

Kay: Yeah.

Lampson: Well, that's kind of appalling. They saved some money somewhere. That's right.

Kay: So let's see. After Harvard, you were fooling around at MIT. You must have had a kind of dual life then.

Lampson: Oh definitely.

Kay: On the other hand I saw in your CV that you got high distinction from graduating. So you must have studied at some point.

Lampson: I studied a lot of physics when I was an undergraduate, that's right. I was only partly hooked on computing. I went to graduate school. Princeton turned me down, fortunately. So I went to Berkeley to study Physics.

Kay: Now you have to tell the great story here of what happened to you on your way towards studying physics.

Lampson: You mean about Steve Russell?

Kay: Just go ahead and tell that whole sequence, because it's a wonderful story.

Lampson: Well, there I was at Berkeley, pretty much disconnected from computing because the Berkeley computer center was not that heavy-duty at the time. But it happened that the 1964 Fall Joint Computer Conference was in San Francisco, so I went to it. I ran across Steve Russell, who I'm not quite sure what he was doing out there -- maybe he was at Stanford -- but he'd been at MIT. He asked me what Peter Deutsch was doing. I had never heard of Peter Deutsch. So I said, "What are you talking about?" Then he explained to me that if you went into the southeast corner of Cory Hall on the first floor, there was an unmarked door. If you went through it then there would be a little lobby, and you could go through another door and then you'd be in a big room. Then if you went through a third door, you would be in the room that housed the so-called Genie Project, although perhaps it wasn't called that at the time. Peter was working on the project. He was an undergraduate at Berkeley. This was all news to me, but I thought it sounded cool. So I did that. I walked into this big room. It was a huge room. In one corner there was a …Bendix G-15? Is that was it was called? Yeah. Right. A Huskey machine, because Harry [Huskey] was a professor at Berkeley at the time. Then in the middle of the room there was this tiny little computer with a guy sitting at it, and he was reading in a paper tape. I stood there and watched, and the paper tape wasn't that long. He read the whole thing. Then he took it out and he wound it up and he read it in again. I said, "What on earth are you doing?" He said, "It's a two pass relocatable loader." I said, "What?" He said, "I know. I know. I'm reloading it." That was my introduction to Peter Deutsch and the [SDS] 940 project, which of course wasn't called that at the time. Shortly after that I gave up physics, because this was really a lot more interesting. It was a good thing, too, because if I'd stayed in physics I would have gotten my degree right about the time of the great late sixties crash in the physics job market.

Kay: Nuclear winter.

Lampson: Right. I guess there's been a couple of those crashes in the last few decades, but this was the first one.

Kay: Let's see, this is '65 right?

Lampson: Right.

Kay: So Dave Evans was just leaving towards Utah…

Lampson: I guess. At that time I didn't know that he was leaving. He was nominally in charge of this project.

Kay: Nominally in charge.

Lampson: Right, although I think his main role was to get money from ARPA. He didn't actually do much technically, because I think he was already focused on what he was going to do in Utah. But us lowly students didn't know anything about that at the time.

Kay: Well, Dave was good at letting people do work.

Lampson: That's right. That was what he did.

Kay: And Harry Huskey is a really nice guy.

Lampson: Harry was very nice, although his role was purely symbolic as far as I could tell.

Kay: You once said that Harry Huskey let you get away with murder.

Lampson: Well, yeah. He was nominally my thesis advisor. He paid absolutely no attention to what I was doing. <Laughs> Then I guess two or three years later he went off to Santa Cruz to sort of retire.

Kay: There was another character there, Mel Pirtle.

Lampson: Right. He was the actual leader of the project. He was a hardware engineer.

Kay: But you and he really became partners a bit in that, because you were starting to concentrate on the operating system.

Lampson: That's right. Peter and I did the most of the operating system work, and Mel did all the hardware design and also ran the project.

Kay: A friend of mine in graduate school, Steve Carr, who was there at that time, told me a story which was hard to believe. One of the interesting things about the Genie project was that it was so well documented, comparatively.

Lampson: Well, okay.

Kay: A lot of the documentation was written by you. Steve Carr said that they couldn't get anybody to write the documentation, and they tried to get you to dictate it. So did they actually sit you in a room and…

Lampson: No, that's not actually true. What is true is that we had to write a progress report at one point, and I dictated the entire progress report. It was the only time, in the days when dictation was actually done to people, that I ever did any substantial amount of dictation. I dictated this 30-page progress report. I thought it came out amazingly well, since I'd never done any dictation before.

Kay: Yeah. Well, I did too. The next year, maybe it was the same year.

Lampson: Maybe it was.

Kay: 1966.

Lampson: But no, the technical documentation was not done that way.

Kay: In my brushes with you, and then working with you, one of the analogies I've always made is a little bit towards Bertrand Russell tendencies.

Lampson: Heavens. In what sense?

Kay: Well, here's one of them. It's quite an amazing fact that all of his popular books, which were exceedingly well written --he had 65 of them -- were actually dictated. He dictated at the rate of 3,000 words a day...

Lampson: Okay.

Kay: …from front to back, and he was done. He made his living writing these books.

Lampson: Well, I never did that.

Kay: I was looking for a kind of Bertrand Russell character when I first met you.

Lampson: I never did that. This progress report was the only thing that I ever dictated until I started using the computer for dictation. Now I tend to use dictation for any substantial piece of writing I do, because although I type very fast, I make a lot of errors. So it's actually quite a bit faster for me to dictate and correct the errors that the recognizer makes than it is to type and correct the errors that I make. But in between, as far as I know, I never did any dictation. The only other substantial thing was that in the summer of 1966 there was an investment banking firm called White Well in New York City. They were a white shoe investment banking firm subsequently acquired by Merrill Lynch. They had an up-and-coming young manager who had talked them into the proposition that he should develop an online stock market analysis system. And…

Kay: Still a dream.

Lampson: Well…

Kay: People do 'em.

Lampson: People do a lot of them. That's right. And they're basically not that different from the thing that these guys built in 1966 and '67. They bought a [SDS] 940. I think they bought [the serial] number 3 or number 4 940, because it was really the only general-purpose time sharing system that you could buy at the time. He had some people who were developing his so-called first financial language. He hired me as a consultant for a month to teach these guys all about the 940. I gave a series of lectures, and they were transcribed, and the transcription was edited a fair amount. That wasn't exactly dictation because I was actually lecturing to a group. But it was close.

Kay: I think this legend is intact.

Lampson: That's the only other case I can think of.

Kay: Let's talk a little bit about the 940. I thought it was one of the gems of its era, which had very few gems actually.

Lampson: What do you mean? There was CTSS. Or you think that was an earlier era? That was certainly a gem.

Kay: No, [and] I think the [Burroughs] B1000 is a gem.

Lampson: Okay. But CTSS was certainly a gem too.

Kay: It was in what it accomplished.

Lampson: Yes.

Kay: But the 940 is interesting in <u>how</u> it did what it did, as well as <u>what</u> it did, I think. Because, I guess through happenstance, the project started with a machine that was probably less than they would have wanted to start with if they had their druthers.

Lampson: They would have liked to have had a PDP-6, I think, but they were extremely fortunately that they couldn't afford it.

Kay: So they had this 24-bit [SDS] 930…

Lampson: Down the hall, you know, we had Don Glaser, who had won the Nobel Prize in Physics for inventing -- the bubble chamber wasn't it?

Kay: Bubble chamber.

Lampson: Right. He had switched to biology, and he was building an automated cell culture lab. He bought a PDP-6 to run it, because he had quite a bit more money than we did. He was extremely unfortunate, because one of the memory boxes of his PDP-6 was dropped several feet off a truck. Unfortunately for him, DEC was able to repair it. But of course it never worked right after that. So he had terrible reliability problems with his machine. The PDP-6 was not a reliable machine in general.

Kay: Anyway, ... cold solder joints….

Lampson: This was much worse. Also apparently somebody explained to me that they didn't actually finish the circuit design for the circuits in the PDP-6. They only got it about 90 percent done, and then they ran out of money and they said we're going to ship it anyway.

Kay: I think that true. They had one at the Stanford AI project.

Lampson: Right. Very flaky.

Kay: Pretty flaky.

Lampson: So anyway, we didn't have a PDP-6. We had this SDS 930, which was a rock solid machine because apparently the SDS circuit designers were really first class. In fact, Mel told me that a year or two later he was teaching the Digital Circuits Design course at Berkeley, and he loved going through the usual academic stuff about flip-flops and gates and this and that. At the end he thought that he should show them how it's really done, so he got out the drawings for the top-of–the-line SDS flip-flop module. Because SDS, like DEC, started out as a module company and they still sold a lot of modules and people would build instrumentation to hook up to their computers. Mel said he spent two weeks trying to understand how the thing worked, and failed. He could not figure out what all those components were actually doing.

Kay: It's like one of these old McIntosh high-fi amplifiers.

Lampson: That's right. It was exactly like that.

Kay: No matter what happened it was not the same.

Lampson: That's right. The 930 was as solid as a rock. It was just great.

Kay: It's a 24-bit machine.

Lampson: That's right. They started out with the 910 and 920, which were 24-bit serial machines, and then they wanted a faster one. But they didn't want to make it parallel, so what they did was made it 3-bit serial. It processed the data one 3-bit nibble at a time, so there were 8 clocks instead of 24.

Kay: And the plan was to put a memory mapping unit on it, to do timesharing.

Lampson: That's right. This was Mel's vision. He was inspired by the [Manchester] Atlas, but this was on a much smaller scale than the Atlas. The 930 had a 16K address space, and it was divided up into 8 2K pages. So we had a little map that had 8 entries, one for each of the 8 2K pages, and each entry was something like 6 bits, because we didn't have very much real memory either. It was all implemented in flip-flops. You couldn't buy memories in those days.

Kay: And in the operating system, you did this kind of fun-- tell me if I'm wrong here, but my recollection…

Lampson: Which might be just as good as mine by now.

Kay: Well, let's make it a good story then. My recollection is that-- The machine had a couple of -- this is judging from the Englebart version of this machine -- it had a couple of interesting things. One is, it was designed as though it would occasionally crash. One of the things that was true on that machine, compared to almost any other time sharing machine beforehand and for quite a while afterwards, was it would generally come back very quickly, because you spent some of the cycles writing out pages to make them clean, as I recall. The idea was that…

Lampson: Maybe. I don't remember anything about that aspect of it.

Kay: Well, I do. Like when I was talking to Ivan Sutherland about Sketchpad -- he had forgotten it because of course he'd gone onto something else, and I was doomed to remember it. But you did this interesting thing, which was, first, you didn't want to swap anybody out in order to bring somebody in. So you did have a map [knack?] for cleaning dirty [pages].

Lampson: That's right.

Kay: And you did scurry around writing out…

Butler Lampson: Writing out dirty pages?

Kay: So you would have clean ones to swap in.

Lampson: Well that makes sense. If we did that I'm sure it wasn't for reliability. It was just so that we would have clean pages in case we needed them.

Kay: But then it had this nice thing that when a crash occurred, lots of people were safe out on the disk.

Lampson: That part I don't remember at all. So you think at after it rebooted it would pay attention to that?

Kay: I think so.

Lampson: Wow. That's impressive.

Kay: I used that idea in my thesis later. And I know I got it from you. I'm pretty sure I didn't make it up.

Lampson: All right.

Kay: Although I'm accused of making up ideas.

Lampson: Well, I don't remember anything about that.

Kay: Another interesting thing was because the address space was small, you could easily fit more than one user in, with their working set, in memory. That was pretty nice because you could have them kind of lined up, three or four in a row, in the Englebart case.

Lampson: That's right. And you could share pages between the users, so when you wrote a program that was intended for heavy use you would carefully design it so that all the pages that were going to be the same for all the users -- all the stuff that was going to be the same for all the users -- would be segregated onto its own pages. Then you would carefully set this sharing up so that if you ran the editor, for example, you only needed one 2K data page. That's six kilobytes <laughs> for each user. And you could do a lot. Well, you know, I used to think that we were amazingly economical until I talked to Rod Brooks about the Roomba vacuum cleaner. He informed me that the computing budget for the original Roomba was 40 cents. Because, he said, it sold for $200, which meant that it had to be in the container

at the factory for $40. And it has some expensive components like a battery, so there just wasn't much money left over for computing. He said it has 10 or 15 kilobytes of ROM and -- get this -- 256 bytes of RAM.

Kay: It's like that old Algol compiler that, you know, had something like 60 passes.

Lampson: That was a Fortran compiler.

Kay: Was it a Fortran compiler?

Lampson: Yeah. That was Lenny Haines' Fortran compiler for the [IBM] 1401. Because his observation… [The] 1401 didn't have disks at the time, at least standard ones didn't. But it had very good tape units. So his idea was just keep the compiler on it…

Kay: That was my first machine also.

Lampson: The 1401?

Kay: We had 8K 6-bit bytes and we managed to write a little operating system that fit in the top 1K for doing things. It did have very good tape drives.

Lampson: Right. So the idea was we had all 60 passes lined up on the one tape drive and you just read them in, choo choo choo choo choo. You kind of messed around a little bit with the code in each of the 60 passes. By the end it was magically converted from Fortran into….

Kay: Okay. The last sort of inner question about the 940 was: another thing I recall is that if a programmer could predict to the operating system what the working set was going to be in the next time quantum...

Lampson: Well, there was no real concept of working set—"on the next"-- you mean what it-- what it was going to change the map to? Yeah. I think that's probably right. Because we didn't actually have any…

Kay: You could punish the programmers if they…

Lampson: Predicted wrong?

Kay: Yes.

Lampson: That I don't remember anything about.

Kay: You would punish them by running them on the slow queue for a while.

Lampson: That's certainly possible.

Kay: So the Englebart people had gotten really good at predicting their next working set. Because of course they were trying to get subsecond response with 10 or 15 people on a 192K half 24-bit machine. When I show the movies of this thing, I always ask the audience, "Why did these guys get subsecond response?" The answer was "they really wanted it."

Lampson: That's right. Well, the whole 940 was like that. It was the first real time-sharing machine, for example, that had character-by-character interaction, because that was a very high priority for Mel. All the earlier systems and almost all the later ones took the view that you would type in a whole line and hit return, and the system wouldn't pay any attention to you until you hit the return. I remember in the APL system the goal was -- that had not teletypes but IBM Selectrics as the terminals, and they had rather slow carriage return -- the goal of the original APL implementation was that by the time the ball made it back to the left edge they would be ready to print the answer.

Kay: I'm going to ask you what you think about that [940] system. My opinion of that system was it was one of the… It fit into my aesthetic space like the LINC and the [Burroughs] 5000 did, and that it was just really a gem.

Lampson: I think that's exactly right. And I think the credit for that, almost all of it, goes to Mel, because he was the one that had the vision of what you should do and the scale at which you should do it. He basically laid out the whole structure of things within which the work at Peter and I and others did was able to fit. But, you know, we were too young and naïve to do that kind of high level design. At the time it was really Mel that did it. And I don't know, maybe Dave Evans made some contributions to that too, but that was out of my vision.

Kay: He didn't claim… I think he was interested in protection, the matrix protection of schemes in the clear [???]. He was interested in interprocess communication kinds of ideas. But he was already thinking about 3D graphics…

Lampson: Right.

Kay: …stuff at that point. But anyways, that's… You know, in a museum I would have one of those big rooms with computers in a museum for gems as opposed to [???] you know, leftover hardware. The 940 is one of these things where it was just a great mixture of design and aspiration. Basically you got about as much out of hardware and software as anybody could, starting from that base, I think.

Lampson: Yeah. I think that's right. And it wasn't just us that did that. Other people like Engelbart and the folks at BBN working on LISP.

Kay: Those are your perfect users in a sense.

Lampson: That's right.

Kay: Because they were really good themselves, and they were able to take matters into their own hands as far as what their system was going to be, and it meshed well. And Peter was kind of an intermediary.

Lampson: That's right.

Kay: Wrote part of his file system for them for the special… But that was kind of a neat…

Lampson: Yeah. Peter and I both worked for Doug.

Kay: Oh you did?

Lampson: Yeah.

Kay: I didn't realize that.

Lampson: Yeah. In the days before Doug got his 940, he had a CDC 3100 that he was prototyping the oNLine System [NLS] on. I wrote about two thirds of the SNOBOL system. It was never finished because the summer ended.

Kay: This was SNOBOL III?

Lampson: Yeah, it was SNOBOL III. Right. Then later we wrote a full fledged SNOBOL 4 system for the 940.

Kay: Then I have to tell a Butler story. Which is: This would be probably in '69, before the ARPAnet starting working. It was one of these meetings that was held periodically. This meeting was at Utah. It was one of these all-day meetings for different considerations of the ARPAnet. At this ARPAnet meeting the usual suspects sat around, debated this and that. At the very end of thing you had been slated to give a presentation about the new Cal TSS...

Lampson: That's right.

Kay: …system that you were doing. Things dragged on and you kept on looking at your watch, and finally you only had 15 minutes before you had to go to the airport. So you gave this entire hour presentation in 15 minutes. It was essentially what was written in this paper you wrote called "An Overview of Cal TSS" which is unfortunately not available on your site.

Lampson: Well, I have it.

Kay: I know. I do too. But we should…

Lampson: I haven't updated my site for quite a while, and I should do something about that.

Kay: This was just an absolutely brilliant thing, and perfectly delivered just at four times [normal speed]. You got basically everything in this hour presentation in there, and dashed off to the plane.

Lampson: Well, I don't remember the 15-minute part. I remember giving that talk.

Kay: After you dashed out the door, Steve Carr turned to me and said, "You know, it's a privilege just to know somebody like that." So, talk a little bit about Cal. I could say that this particular piece of work, this particular paper, was seminal for thoughts I was having at that time. It really helped me look at the B5000 and the object stuff I'd been doing and the larger operating system. I mean, there's one of these interesting things where the mechanisms in there, if you could handle different scales in a reasonable way, were kind of universal in the kind of coverage they do. I would call them, compared to the way most operating systems do their thing today, I would call them pretty darn modern.

Lampson: I think that's fair.

Kay: So say a little bit about that.

Lampson: Well that's, see, the…

Kay: Did that come by way of Multics?

Lampson: Not all at. No. I mean, I knew about Multics.

Kay: Because they didn't have a flag bit, they would have whole files full of capabilities rather than intermixing.

Lampson: More or less. But of course they had much more elaborate hardware support for all this stuff. The way it got started was that the Berkeley Computer Center decided to upgrade from the 7090, or whatever system and they had, and they bought a CDC 6400. But actually they managed to persuade NSF that they should do some research in systems, and so the NSF ponied up for two 6400s. The second one was mostly dedicated to the development of this Cal TSS system.

Kay: Also, you got this humungous memory.

Lampson: Well, it was weird because it wasn't really memory. It was this thing called extended core, which was like a disk in the sense that you couldn't execute programs out of it, and although could reference data there you had to use a different instruction set in order to do that, and it was relatively slow. Mainly what we used it for was as a very high speed swapping device. So we had not very much main memory at all. I think we had 32K of 60-bit words of main memory, and then we had half a million 60 bits words of this so-called extended core.

Kay: You started this overview of this system with a great paragraph about the bitter experience that operating system designers have gone through, and the people who are forced to use the results of their…

Lampson: Yeah. Generally things have been pretty tough, and I think things were pretty tough for the users of this system too. Because we started at ground zero here, and we did manage to get it up to the point where it had users. But by that time we were getting kind of tired. <Laughs> So in that respect it was quite a bit different than the 940 system, where really I think most of the effort went to the applications. Although we did have to start from ground zero.

Kay: But you had this notion that you were going to use confinement and…

Lampson: Well, we had the idea that we were going to base the whole thing around this idea of capabilities. Everything was going to be accessed through a capability on this. In that sense it was a tremendous amount of uniformity in the system design. I'm sure that was the thing that influenced you.

Kay: Reference table technique.

Lampson: That's right. Exactly like that, but somewhat generalized.

Kay: That's one of the great ideas of all time, I think.

Lampson: Well, it's interesting because certainly this capability idea has been around for a while. We didn't think it up. I think it was Bob Fabry that originally articulated it and gave it that name. Although, as

you say, the roots of it go back a lot farther, to things like the B5000. But I think it's really fair to say that it's been a bust.

Kay: Why?

Lampson: Well, I think it's because people have had a mixed set of goals, for the most part, when they built at least systems that they called capability systems. Usually a very heavy focus of it -- and this was certainly true to a considerable expense for us -- a very heavy focus of capability-based systems has been on the security aspects of it. That's not a good focus to have when you're building a system, because fundamentally the users don't care about security. Right? They want to get their work done. Offered a tradeoff between an insecure system that let's them get their work done, and a secure system that doesn't, they will unhesitatingly choose the first one. So I think people have gotten very much seduced by the security aspects of capability systems. For security, actually, capabilities are not that good; the reason being that they put the focus in the wrong place. They put the focus on the source of requests to do things, because typically the way capability system works is: I've got a capability, and what that means is that I can use it to do something. But that's not what you want, typically, in security. What you want is for the resource to be able to protect itself. Capability systems don't do that. You can twist 'em around, of course; anything can be twisted into anything else in computing. But it doesn't twist very comfortably.

Kay: You could think of acute [?] capability, or extended capabilities, kind of like a view on an object.

Lampson: That's right, which is fine for programming. It's not so good for security unless you put in more levels of indirection, which people typically don't do because they're all so focused on their… So a bunch of people have built capability systems. Cal TSS, I believe, was the first actual working capability-based system. A number of people have built capability systems with a lot of hardware support, most notable Cambridge.

Kay: Yeah. Needham.

Lampson: Needham and Wilkes built the Cambridge CAP computer. They've all been failures. People are still working on it. There are these folks at Penn -- Jonathan Shapiro, is that his name? -- and his colleagues have this EROS system, or they keep renaming it and I'm not exactly sure what it's called today. And they still passionately believe in it, but I don't believe in it at all.

Kay: <Inaudible>

Lampson: <Laughs> So Cal TSS was really interesting. There was a lot of interesting intellectual stuff in there. By the way, some of which came from me, but the majority of it came from Howard Sturgis.

Kay: Howard, yeah. So a digression here is…

Lampson: And, you know, Cal TSS was the first system that had transactional files. They didn't…

Kay: I didn't know it was the first, but I remember…

Lampson: We didn't actually use them for—well, there were probably database systems that had some semblance of transactions at the same time, or maybe even earlier.

Kay: So <inaudible> was a graduate student?

Lampson: That's right. That's where he first learned about this stuff.

Kay: Because I remember I came up and gave a much discussed and criticized talk about the Flex machine to some group of your graduate students.

Lampson: That's right. Jim Gray was a graduate student. Charles Simonyi also worked on Cal TSS before he went off to [Xerox] PARC and then later Microsoft and became a billionaire.

Kay: So every bacterium that I might have had on my skin had been killed.

Lampson: Well you have to realize, you know, Jim Gray wrote his thesis on automata theory. And Howard Sturgis at the time was best known for having written a paper jointly with the fellow from England named Shepherdson who'd been visiting Berkeley for a year, and they basically showed how you could prove theorems about the capabilities of Turing machines much more easily if you started out by making a Turing machine simulate a computer. Then you would prove theorems about the computer, and since the simulation didn't have any interesting complexity properties the theorems were equally good.

Kay: Turnabout is fair play. Okay here's a little digression, which is: I've often wondered what would have happened in functional programming if that field had had a Dan Ingalls. This brings us around to an interesting thing; between good ideas, and that it often takes a special person or a small group of people to actually make a medium-to-good idea work. Sometimes you never know whether the idea is good or not until it actually gets done.

Lampson: Well, functional programming actually has been extremely successful in certain limited domains.

Kay: Yeah. I was just using it as a-- I'm just saying that in…

Lampson: I don't think it would have helped to have a Dan Ingalls, is what I'm trying to say. Because I think… I mean, the things that… The paradigm that Dan was working with and the Smalltalk paradigm

really is general purpose. Functional programming -- if it's general purpose, we haven't seen it yet. Even now we don't see it. It works great for things like SQL. It worked great for APL, which was functional except at the topmost level. It works great for things like Matlab, and Mathematica, and Maple. It works well for spreadsheets. But those are all domains that are really quite circumscribed. I think it continues to be the case that we really don't know how to make functional programming work for general purpose.

Kay: [Do you know about] Lucid?

Lampson: A little bit, yeah. Not deeply.

Kay: Lucid was a way of looking at functional programming. I always felt that if they had a Dan Ingalls…

Lampson: They could have made it fly?

Kay: They could have made it fly.

Lampson: Well, it's been very seductive. A lot of people -- a lot of smart people -- have tried it.

Kay: I've tried to get up with <inaudible> capabilities because they're all smart people all out there. But we also had the privilege of working with kind of ironman programmers, luckily, over our careers. And they can make…

Lampson: But we couldn't make capabilities work.

Kay: Yeah. But maybe if you had to do it again... I don't know.

Lampson: Well, that's possible. It's definitely not how we felt. Nobody who worked on that project wanted to build another capability system, even Howard, who's a very principled kind of guy. Nobody wanted to do that. So I'm very skeptical. No, I really think that it's a red herring.

Kay: So moving to Xerox PARC now…

Lampson: Which presumably is what they actually want to hear about. Skip over BCC [Berkeley Computer Company].

Kay: Skip over BCC a little bit. But when, I guess, you and Taylor thought BCC was faltering, and Taylor convinced you and eight or nine of your very talented colleagues…

Lampson: Yeah. That's right.

Kay: …to come to over to Xerox PARC; the dawn of Xerox PARC.

Lampson: That's right.

Kay: I was consulting for Taylor at the time; there were maybe three or four people there. So that was a huge change in January or February of 1971 there.

Lampson: That's right.

Kay: A bunch of interesting things happened within the first month of two. One was a story I'm going to take up in a second, which is the Maxc [Multiple Access Xerox Computer] story. But another was kind of a philosophical thing that I have a very clear though possibly false memory of, because I told this story so many times that I'm sure I've done something to it. It had to do with what I've characterized as an argument you made to all of us, that we essentially should not do anything at PARC that wasn't really engineered for a hundred users. The angry part of the argument, not directed at us but just sort of directed at the heavens in general, was basically: goddammit, you know, we sort of spent the sixties doing more bubble gum and bailing wire kinds of things, and it's hard to expand…

Lampson: Well, now wait a minute. We didn't do that on the 940 system. We built that for a hundred users.

Kay: So maybe you were directing it at us. The Flex machine is certainly not…

Lampson: Well, yes.

Kay: We were lucky that what parts of it were that worked.

Lampson: That's right.

Kay: But anyway, that was something that to me at least, and I think to a bunch of us, was really a turning point, of thinking about this next phase of the life of this thing, and trying to do a little more engineering earlier. It had this really interesting effect in a lot of things. I felt that a lot of things got done rather more quickly because people were "taking more time" early on with the engineering aspects. They somehow seemed to get done more quickly. Does that jive at all with your memory? Like, you know, Maxc was quite a feat.

Lampson: I think that's true.

Kay: I mean it got done pretty much when you guys said it was going to get done, and the Alto got done in a much faster fashion than that.

Lampson: Of course Maxc -- there was no research. There was really no uncertainty about Maxc. That was fairly straightforward engineering job.

Kay: Yeah. There was a lot of stuff…

Lampson: There was a lot of stuff to do.

Kay: A chip that had did not test all that well from Intel so…

Lampson: Oh, it didn't work.

Kay: Didn't work. Right.

Lampson: But we thought it probably wouldn't work. That was okay. We allowed for that.

Kay: Yeah.

Lampson: We used some other Intel chips that worked fine.

Kay: This is probably the first mainframe computer to actually use integrated circuit memory.

Lampson: I believe that's true. I think for a while we had the world's largest semiconductor memory. We had three megabytes. It was in four six-foot racks, all built out of these one kilobit 1103s that didn't actually work.

Kay: And Dick Shoup had made, I think, two testing machines: one for PARC and one to give to Intel so they could test their own chips.

Lampson: No, it wasn't because of that. You don't know that story? We were buying these 1103s from Intel, and it was great, and we had our test setup. Then one day Intel called us up and said, "We're not going to sell you anymore 1103s." We said, "What?" Now I'm going to embroider a little bit, but Intel said, "We want to become a systems company. We want to make your boards. We don't want to sell you chips." So we said, "Oh shit." Then we said, "Well, okay. We can give you our test setup and explain to you how we test these things and so forth, and I guess it's okay if you make the boards." So

we gave them all the stuff that we were using to make the boards, and they made the boards, and it was kind of okay. But no, they only used them to test our boards.

Kay: …pushed a little further.

Lampson: Well Intel has made…

Kay: I was surprised at how daintily they put their foot into the waters of computing.

Lampson: Oh, on the contrary! They put their foot in with a huge splash and they butcher it every time.

Kay: Oh oh.

Lampson: Remember the [Intel iAPX] 432 [32-bit microprocessor]. There was nothing dainty about that.

Kay: That was later.

Lampson: Not that much later.

Kay: But I think everybody messes up on a project like the 432, the first… This is one where I think it could be done twice. Don't you think?

Lampson: No. Doing the 432 twice wouldn't have helped at all.

Kay: That reminded me so much of the all the over-optimization that Cheadle and I tried to do on the Flex machine. You know, it was just too…

Lampson: But the fundamental problem with the 432 wasn't over-optimization. The fundamental problem with the 432 was that the grand vision was broken.

Kay: Well maybe the object model that they were shooting at was not a very good <inaudible>.

Lampson: I think that's exactly right.

Kay: Just do a Hydra kind of object model.

Lampson: I think that's exactly right.

Kay: A big object [model] rather than a fine-grained object model.

Lampson: Well, I think they actually… No, I would have said it the other way around. Actually I think they fell into the same trap that the [Cambridge] CAP machine and a number of other people have fallen into, which is they've said, "We're going to use the fact that we control the hardware to give you hardware-supported fine-grained access, but of course only of particular limited kinds, because they are the only kinds the hardware knows how to support." Typically you pay a huge amount for that in hardware resources and performance. And what happens when you try to use it, is you discover it's not flexible enough. So you end up having to interact through the software anyway, which means all the hardware does is get in your way.

Kay: To me that was…

Lampson: You never had that problem because you never built any hardware, right?

Kay: Well, the hardware got built because it was done by Cheatle, but he listened to me too much.

Lampson: Okay.

Kay: The way I look at the thing is… The thing that I learned from you and Chuck [Thacker] is: don't basically try and use your microcode as an escape route as well as a power. Try and leave as many solutions away from the hardware. Of course the Flex machine was a lot slower, so we had to think a lot harder. But basically I would say the Flex machine tried to optimize way too many things in hardware. That reminds me very much of 432.

Lampson: Yeah, but I don't think that was the fundamental problem. This comes back right back to what you were saying before about the hundred users. I think the biggest payoff, from my perspective, that we got from setting that as a goal was that it prevented us from setting the aspiration level too high. If you contrast the work that was done on the 940, or the work that was done at PARC, with Multics for example -- the Multics guys just had much more ambitious goals than we ever aspired to, in terms of the conceptual integrity and completeness of the system and gloriousness of the system they were building. I think all of our experience with this is that it's very easy for that to spiral out of control, and in fact it's almost impossible to keep it from spiraling out of control.

Kay: Well, that's why I thought that the… I've often characterized PARC to people, and they're surprised, that I've said, "Well, it was more engineering-like than you would think."

Lampson: Absolutely.

Kay: In the sense that most of the ideas were things that various of us, and other people, have thought up in the sixties, and they're trying to figure out how to make them really work in some reasonable scale size and also something that would scale …

Lampson: Right, exactly. You know, between the LINC and Sketchpad and CTSS and a few other things like that that were done in the sixties, you could make a perfectly good argument that there was no new idea in what was done at PARC at all. I think that argument is pretty bogus, but you could certainly make it fairly convincingly. I think this was a direct outgrowth of the fact that most of us tended to take the view that we weren't going to embark on something unless we could see a pretty clear path to actually engineering it so that a bunch of people could use it.

Kay: Also, I don't think there was any "thesis lure".

Lampson: Well of course…

Kay: Particularly at PARC.

Lampson: Right.

Kay: We had already done that, so we didn't have to be that creative in that sense.

Lampson: Right. I have a clear memory that in the summer, I think it was in 1974, I had come down with hepatitis and I was stuck at home for a couple of months. A couple of times people came over and we had discussions about how the Alto software should evolve, because the Altos were just starting to do interesting things at the time. And at least from… I mean, the Smalltalk group took a somewhat different tack on things. But from the perspective of CSL, I was arguing very strongly against any attempt to build a wonderful glorious unified system. Because, I said, I don't think that the hardware resources will support it. I'm confident that we could write a bunch of code that would have that effect, but the net result would be a system that would be too sluggish and too painful to develop, and we absolutely shouldn't do that. Instead, we should use the wonderful world-swap idea to switch between different worlds and let people do their own stuff and really provide them with most minimal overall support.

Kay: Yeah. In fact one of the things I characterize PARC as being really unusual, in that given all those strong personalities there -- that I think is probably an understatement -- the amount of tolerance there is pretty darn amazing.

Lampson: Mm-Hmm. Well, a lot of that of course was deliberately engineered.

Kay: Yes. But even so, that tolerance allowed two or three mainstream things that had somewhat different aesthetic centers aiming at the same things. I think in many other institutions our little group,

which Taylor used to call the ”lunatic fringe”, would have been quashed, because you’re not following the party line. But instead I felt that we were not just tolerated, but people helped whenever they could just to see what we were going to do with this more ambitious… And on the Alto [it] was really tricky to have just 80K bytes to try and swap onto this little model 30 disk drive.

Lampson: That’s right. That’s right.

Kay: But system-wise, it was a really interesting thing to do in parallel with what you guys were doing. There’s a lot of going back and forth in user interface ideas, and arguments, and everything. I think that to the extent that you, and also especially Bob Taylor… This is good time to bring up Bob. When did you first meet Bob?

Lampson: Well, I don’t have a clear memory of this, but I must have run across him when I was at Berkeley on the Genie project and he was running ARPA,, because he was Ivan Sutherland’s deputy for a while. Then he was running ARPA for two or three years, and we were getting our money from them. So I’m sure I met him then, but I don’t have any clear memory of that.

Kay: There was an ARPA principal investigators meeting at Utah that Dave invited his graduate students to, and you and Mel [Pirtle] were there at that time. So it would have been 1968, I think.

Lampson: Sounds plausible.

Kay: You’ve known Taylor for a long time, and in different guises.

Lampson: Of course I only started to work with him closely after he recruited us to PARC.

Kay: Yeah. So what did he say to you to get you to go there?

Lampson: I don’t have any clear memory of the words, but I’m sure that the content was: BCC’s coming apart, and you’re probably planning to go back to Berkeley, which was true. This is great opportunity to do something that will be a lot more exciting. It seemed fairly convincing. There was Xerox. They had said we’re going to start this new lab. You’re job is to invent the office of the future. We don’t have any idea what that means, but you figure that out. How could you go wrong? The reason we started BCC was that we wanted to build a follow-on system to the 940. We learned after about a year and a half of futzing around that it was essentially impossible to do that at Berkeley, because there were too many organizational constraints imposed, not so much by ARPA but by the structure of the university. I’m sure if I’d gone back to Berkeley I could have done interesting stuff. But from the point of view of that experience, PARC seemed like a great opportunity. In 1969, if we’d been able to go to PARC and build the BCC system instead of starting a company, we’d have lept at the opportunity to do that. We didn’t start BCC because we wanted to make a lot of money. We started BCC because it was the only way we could think of to carry out this particular research program. And of course, the venture capitalists should

have realized that that was not a good basis on which to fund a company. <Laughs> But our motivation was perfectly clear. At least mine was.

Kay: That brings up one of those more meta-existential type questions, which is: you must have, at some point, especially when you switched out of physics, you could smell a different perfume, or feel a different magnetic field pulling at you.

Lampson: Yeah.

Kay: And every once in a while you must have thought about it. What it was that… what's so neat about… you know, what…

Lampson: What's so neat about computers?

Kay: Yeah. And what, well, what's so neat about what the next thing is going to be? In the sense of… I mean, you've always had a drive for not just interactive computing, although it's been a steady theme throughout your career. But you really have been a major contributor in lots of different areas: systems areas in general, programming languages, networking. I think of you as kind of the quintessential computer scientist, you know. I don't know what computer science means, exactly, but I think you are a computer scientist. You've basically done most of these things in this peculiar combination of science, math, engineering, and the fact that you have to build systems to know whether what your claims are mean anything. But there's also an aesthetic component which I know has always drawn you.

Lampson: Of course aesthetics has been important. I suppose that by the time we were starting PARC some of us including myself were starting to have some kind of grand vision. But that wasn't the case in the 1960s. It was interesting engineering, was what it was in the sixties. That was perfectly clear. It was an opportunity to do these really cool things because you had this wonderful malleable medium that you could work in.

Kay: So the "PDP-10 in a suitcase" thing came out at one of the -- maybe the last -- Alto conference that we ever did. They used to have these conferences once or twice a year.

Lampson: Yeah, I remember those. I remember there was one where you tried to explain Simula to me. And you failed miserably.

Kay: I could do a better job now.

Lampson: It was not until several years later that I had any clue what you'd been talking about. And I think it might have been a year or two later before you had any clue what you'd been talking about.

Kay: Well I think it's… I would say that my M.O. has been perfuming the way I was…

Lampson: Yeah. I think that's right. You have to decide what you think is going to be important even if you don't see your way clear to it.

Kay: Exactly.

Lampson: And then you have to push on it. And then the question is: Is your judgment good or not? Because there's certainly a lot of ways to go wrong. We were talking about the 432; I'm sure the 432 was, you know -- people smelled the perfume, and they went after it. They just didn't have good judgment.

Kay: Yeah.

Lampson: And with the Multics guys there was no question about that.

Kay: I've avoided AI my entire career because I've never had even one of those perfumed moments about it. I couldn't see something pulling us out of the mist that might actually do something.

Lampson: AI is so funny. You know the story about why AI is always in such a mess?

Kay: Yeah. It's because as soon as you learn to do something you go [into] standard engineering.

Lampson: You spin it out.

Kay: Yeah.

Lampson: It's not AI anymore, right. It's the same reason that philosophy is such a mess.

Kay: Exactly.

Lampson: First you spin out math and physics, and then you spin out biology, and you spin out linguistics and computer science. Whatever it is.

Kay: When you came to PARC you certainly weren't intending to do Maxc, because there is an intent to buy a PDP-10 and that was sort of what precipitated the Maxc?

Lampson: Well, I guess "intended"; mumble. I mean, we knew there were going to be lot of problems associated with buying a PDP-10. I think a better way to say it is that we were confident that we could work our way through this one way or another, but certainly at the time we went there we didn't have any clear vision of exactly how we were going to do that. Of course Xerox wanted us to buy a Sigma because they just bought XDS [formerly SDS, Scientific Data Systems] .

Kay: Right. And doing a new operating system on it would have been…

Lampson: Well, doing a new operating system would have been a lot of work. But it would have been much worse than that, because having done that we would have been cut off from the rest of research community. That was the real killer in it. That was the thing that… I mean, probably we could have done Maxc without this. But that was the thing that basically sold the management on the fact that this was a sensible thing to do. They said, "show us the financial justification", and we made this beautiful table where we added up the hardware costs of this and that, and then we looked at the software costs, and we said, "How much would it cost to reproduce Tenex, and Interlisp, and this, and that, and the other thing." Those are big numbers. Down at the bottom you have two big numbers, but one of them is much bigger than the other one because there's that huge software cost. All of this of course was ignoring the fact that we never could have actually motivated people to do it.

Kay: No. But also I think it was fortunate that management probably knew nothing about the difficulties of doing a computer from scratch.

Lampson: Oh. Well you know, this was right about at the time when it was no longer that hard. If it had been even a year earlier, it would have been much harder. But we had 1103s, so we didn't have to struggle with core memories. The other thing that was equally important was that Intel was making a 16 by 4 high-speed RAM chip, which we used to implement the registers and the mapping [tables].

Kay: No, I didn't mean from the PARC point of view. I meant because projects even with the best hardware available sometimes become black holes.

Lampson: Well that's true.

Kay: And this time it didn't.

Lampson: It could have been a disaster. That's certainly true.

Kay: But because it wasn't…

Lampson: But of course it was a much higher risk that if we tried to develop a new operating system for the Sigma that that would have been a disaster.

Kay: The interesting thing about it was that a lot of good things happened because of Maxc.

Lampson: Yep, that's right.

Kay: This is just observing from the peanut gallery over there. It basically coalesced the CSL team, it got a bunch of things… it produced a bunch of ready-made technologies that we…

Lampson: That we subsequently used. That's right.

Kay: … much easier.

Lampson: Well, some of those things went astray a little bit. I don't know if you know the details of the story of the memory system. When we decided to build Maxc out of 1103s, we knew that we didn't really know whether they worked. But we said that's okay. We can put in an error correction, and as long as they work at all, it'll be okay. And indeed it was. We never had any problems with the Maxc memory. Then we built the Alto, [and] we used actually the same boards…

Kay: Yes I remember that.

Lampson: But we didn't think we could afford the error correction because the words were much shorter and it was actually pretty expensive in 1972 to implement error correction out of not very "M" MSI. So we built the Alto just with parity. In the early days of the Alto we didn't have any really interesting software, and it was looked great. Then when we started to run Bravo we started to get parity errors. We were quite baffled, because we were having no problems with the memory on Maxc, and we hadn't had any problems with the Alto for the first six months. So what's going on? Then we learned that, well, 1103s are indeed pattern sensitive, which we could have found out if we'd done a much more exhaustive testing program. But they weren't very pattern sensitive, so it would have been pretty hard. Secondly, if you run… So why weren't we seeing these problems on Maxc? Well, Chuck Thacker, who designed the memory system for Maxc, provided a register that reported at least one corrected error. But nobody ever wrote any programs to read this register, so we had no idea how many errors were being corrected. The answer was, quite a few. We never got double errors, but those we would have known about because those would have been forcibly reported. But that didn't happen. Three, it turns out that if you just run boring minicomputer software you don't get enough different patterns. But when you run Bravo it sprinkles characters all over the screen and that makes lots of interesting patterns. So that was part one of the memory system story. Chuck had to write a very heavy duty random number memory diagnostic that we would run on all of Altos overnight, and then they would report their problems over the Ethernet. The techs would come in at 8:00 in the morning and change the bad chips. I think that was the first use of networking for maintenance, at least that I know about. You know the second half of this story. Two years later we built the Alto II, which was basically the same, but we upgraded some of the implementations to take advantage of things. In particular, you could now get 4 kilobit RAM chips, and you could also get a better MSI for doing error correction. So we re-engineered the memory system, and we weren't [going to] make that mistake again, so this time, error correction. So the Alto II was built and it

was solid as a rock. Two years later it was discovered that in one quarter of the Alto II memory, neither the error correction nor the even the parity worked at all…

Kay: Ouch.

Lampson: …because of a design error. There are two things going on. First of all, the chips were better. Secondly, you have to think about what happens when a bit flips in the memory, okay? If you have parity, the next time you touch that word you're going to find out about the flipped bit. If you don't have parity, most of time it doesn't make any difference, right? If it does make a difference, who gets blamed? Not the hardware. It's the software that gets blamed. It causes some inexplicable crash, but those are happening for lot of other reasons. So, this is why…

Kay: Half of the Alto memory was the display anyway.

Lampson: In which case it definitely didn't matter.

Kay: It didn't matter.

Lampson: Right. This is why PCs don't have parity. It's not for cost reasons. It's because if you get a parity error, Dell gets blamed. Whereas if you just have a flipped bit, either it makes no difference or Microsoft gets blamed. So from Dell's point of view, putting in parity is nuts. It costs more and it exposes them to more trouble and has no upside whatsoever.

Kay: Well, the vagaries of the Alto certainly got us, since we were trying to maintain dynamic objects. We had this interesting problem, and this is where we used some of the 940 swapping ideas around the object basis.

Lampson: Right.

Kay: And doing checkpointing every ten or twenty seconds or so. That worked awfully well because the Alto had some other, I think, heat related-- it certainly had some random crash kinds of properties that had to be sealed off against.

Lampson: Right. Smalltalk certainly stressed it much more heavily than any of the other software. The only other thing that ran on the Alto that I think people really cared about was Bravo, and Bravo had this other recovery technique where it recorded all the keystrokes.

Kay: Yes. Which I thought was more…

Lampson: Which was really a brilliant-- was originally put in I think primarily for debugging. But it was great for error recovery.

Kay: It was great because it actually built you a loyal end user community right from the get-go…

Lampson: Even when the software wasn't that reliable.

Kay: Basically did two things. You printed for people, and you guaranteed recovery up to the last keystroke. That got a lot of people using Bravo, and I think it helped coevolve the design in a way that it was maybe one of the first times when something had been that closely coevolved. Also, you had this nice working relationship with Charles Simonyi and his little group of people doing the thing. That was a very interesting example of kind of incrementally evolving a very useful design. That brings up actually one question I've been wanting to ask you. This must have happened. Here you are, maybe three days ago or so, doing something with a computer. At some point it flashes through your mind that pretty much every single piece of technology that you're using at that moment was something that you and your research community invented.

Lampson: Mm-Hmm. Yeah. Yeah.

Kay: Right? Have you ever had that, where you're just…

Lampson: Oh yes.

Kay: Where it's just kind of a chain [?] of things. You're using Microsoft Word…

Lampson: Oh yes.

Kay: You're using a windowing interface, you're using a bitmap screen, and you're using the Ethernet.

Lampson: All that stuff.

Kay: Blah. Blah. Blah. The laser printer, Internet. All that stuff. When that happens, it's a little fleeting moment. What do you think?

Lampson: It really makes you wonder when there's going to be some substantial advance. The only substantial advance since the days of PARC that I know about is the Web.

Kay: Yeah. And I…

Lampson: Which really is qualitatively different. It's an interesting question why it took so long to happen, which I have a theory about. My theory is that it's entirely a matter of scale. It couldn't happen until the Internet got big enough, because until then it wasn't worth the hassle of organizing your stuff so that it would be accessible to other people. But things got above a certain scale. Then you could find a big enough user community that you actually cared about enough to be willing to do that work. Because from a technical point of view it could have happened ten years earlier, I think. It's just that it wouldn't have paid.

Kay: But I wish that you had been at CERN on a sabbatical when that….

Lampson: I probably would have been a disaster.

Kay: I don't know. But I think you would have made a slightly better…

Lampson: No. No. No. No. No. No. What Tim [Berners-Lee] did was perfect. My view about the web is that it's the great failure of computer systems research. Why did computer systems researchers not invent the web? And I can tell you the answer. It's because it's too simple.

Kay: It is too simple.

Lampson: If I had been there I would have mucked it up. I swear to God. The idea that you're going to make a new TCP connection for every mouse click on a link? Madness! The idea that you're going to have this crusty universal data type called HTML with all those stupid angle brackets? We never would have done that! But those were the things that allowed it to succeed.

Kay: Yeah, to some extent.

Lampson: Absolutely. Not "to some extent". Absolutely. There's some bad consequences,…

Kay: Boy, are there.

Lampson: …but that's too bad. You've got to go with the flow, otherwise it would… No, it would have been a disaster. Never would have worked.

Kay: I think that, well… There's this wonderful line which you have attributed to other people, about [how] you should always have an extra level of indirection.

Lampson: David Wheeler said that.

Kay: David Wheeler. And Dave Evans used to say, "You should always have an extra bit".

Lampson: <Laughs> Well, in some respect that extra level of indirection is a substitute for an extra bit.

Kay: Yeah, exactly. An interesting consequence about the way the web browsers and stuff were done is that in spite of the fact that they could allow safe downloading of executables and confinement in a way so you don't have to go through sysadmins [system administrators] and so forth to do it, it wasn't done.

Lampson: Sure it was. That's what Java's all about.

Kay: Well, not really because it's… I'm talking about being able to download some… So here's a typical problem that a child has. A child finds something interesting. Somebody has done a Logo [program] or something like that. Generally speaking the child cannot get that on their computer at school because the…

Lampson: Well that's not the Web's fault. That's Microsoft's fault. Or Apple's fault.

Kay: But it's a fault in the systems design of… It would have not been made by somebody like you if you'd been in that loop.

Lampson: I hope I would have had the good judgment not to try to fix that.

Kay: Yeah?

Lampson: My opinion is that fixing that is too hard, and it's not worth it. The price you're going to pay for protecting the guy against all the bad things that can happen from this native code that gets downloaded is going to be too high. People are not going to be willing to pay it. The system that doesn't pay it is going to always win out over the one that does. If you want protection, you should do it in some more circumscribed world like a Smalltalk world, or a Java world, or a CLR [Common Language Runtime] world, or a C# world.

Kay: Yeah. Well, the problem is getting so…

Lampson: You know, in principle a system like the Mac OS X or [Windows] NT can provide that kind of protection. But if you look at all the details of what you actually -- the machinery you have to put in place and the ways in which you have to limit communication -- it gets in the way of a lot of other things that you want to do, and the priority is just not high enough. It's not going to fly.

Kay: Let's try one more example on you here. When I was recovering from some dental surgery [I] was leafing through the net and found the Wikipedia article on Logo, which is actually very good and had a whole bunch of logo examples. But I got very upset that I could not run those Logo examples, since I could have run them on an Apple II in 1970.

Lampson: And why couldn't you run them now?

Kay: Because the Wikipedia didn't allow that Logo… That Logo text was in there just as text because the Wikipedia didn't have a type called…

Lampson: What's stopping you from cutting and pasting it into your Logo?

Kay: Then you have to get the Logo interpreter, if you're a kid, from-- and get the system then to allow the things to be…

Lampson: Well you wouldn't have had that problem.

Kay: No. No. What I'm saying, I was just…

Lampson: You said you couldn't run the…

Kay: No. [Imagine] I'm a child. I always think like my youngest…

Lampson: Well, the Wikipedia can certainly have a link that the kid can click on. The question is will they… the administration actually…? It would be much better if that would work. I'm just trying to point out the downsides of it.

Kay: I'll tell you what I did.

Lampson: Yes.

Kay: I decide, okay, I'll write a little Logo interpreter in JavaScript because I can download a JavaScript automatically. So I did that. I made a little Wiki page, and so there's a thing on the net now called logowicki.net that just does this the way you'd like and it has this stuff embedded. Now that got me looking into what the browser actually does, which is to essentially take away your entire computer except for a couple of weak concepts, because you cannot get something stronger than JavaScript in there without this…

Lampson: Going through this rain dance.

Kay: Yeah. That's…

Lampson: Well you can get Java, which is pretty powerful.

Kay: If your browser's been set up for it. It's not there automatically at all.

Lampson: Well it's pretty standard.

Kay: So I'm just saying that…

Lampson: But the issue there is actually interesting, because Microsoft has struggled with this a lot. There definitely is a world within which you can load Java, or C#, or whatever it is, into your browser and run it safely. But it's actually a surprisingly restrictive world. Numerous attempts have been made to make it less restrictive, and they have not been successful, even though in principle it's okay. But if you actually look at all the interfaces that you would like to have access to, and ask for each one of those is it safe, typically the answer is "no".

Kay: Right.

Lampson: That is, if the bad guy can find a way to exploit that interface… There clearly is no law of nature that says you couldn't design one that would be safe. But people have actually tried fairly hard to do that and they failed.

Kay: If you were going to do a new operating system from scratch and not worry about the legacy problems or anything else, don't you think that problem is solvable?

Lampson: Only in the following limited sense. I think there's a fundamental conflict between the desire to have a bunch of programs that can interact with each other in complex ways, and the desire to have security. If I were doing a new system I would do something that's actually also perfectly feasible to do with today's systems. I would say: Okay, we're going to let you have what you should think of as different worlds within this system. Within one of those worlds you can plunk down more or less any code you want and run it at full speed, but its ability to communicate with other worlds is going to be extremely limited. Then you have to recognize the fact that some of those worlds are going to be big, and they're going to have a lot of stuff in them, and the security is not going to be very good there. There's going to be an explicit design decision. The price you're going to pay for having this isolated thing is that it can't communicate very much with anything else. That you can do perfectly well today with virtual machines, and you can even do it with… But people don't want to do it.

Kay: We are corresponding about that.

Lampson: But people don't want to do that. Typically they want more communication.

Kay: Yeah, but seems to me that those are just my supposition that if a person was going to do something from scratch…

Lampson: Well, see, I don't think you need to do it from scratch. I don't think you could do significantly better doing it from scratch. It's actually is being done by the Singularity [Project] folks in Microsoft Research. But my prediction is the way it's going to come out is not going to be significantly different from having separate virtual machines in the cases where you actually want the isolation. That's my prediction, because I think what will happen is that in the normal case… They started with this concept of extremely well-isolated processes and channels and all that stuff -- those ideas that have been around for 30 or 40 years. In real life what's going to happen is that in the normal development environment a process is going to have lots of these channels. It's talking to lots of things that it needs to be able to talk to in order to do the stuff that it wants to do. It's not going to be secure.

Kay: But of course in a sense, if it looked a lot like the Internet internally, where you're just sending HTTP messaging between these virtual machines, which reminds me of some old ideas of yours from a long [time ago]. Then you would actually have something that would be recapitulating something that scales on the outside and is perfectly well confined.

Lampson: It's not so clear when you ship HTTP messages around how well confined it is.

Kay: It's you're basically putting the burden on the receiver not to do anything stupid.

Lampson: That's right. But we know that real life receivers are going to do things that are stupid. What you just said is completely abstract. It's got nothing to do with real life.

Kay: But of course even if [it] does do something stupid, most of the time it's just one machine that crashes.

Lampson: Well that's… Right. Right, but that's because we did such a crappy job of letting the machines communicate. If we were doing a good job of letting the machines communicate, the problem would be much, much worse.

Kay: Well but this brings us to your… One of the lectures I heard you give recently was about how our objects grew up and conquered the world while nobody was looking.

Lampson: Components, it was. Right.

Kay: Right. And after that I have a very different large object theory.

Lampson: Oh?

Kay: But it overlaps a little bit.

<break>.

Kay: Actually next year will be the 40th year since I've known you in person, which is…

Lampson: I guess that's right.

Kay: …kind of frightening. Actually, I should ask you: Lois, your wife, got her undergraduate degree in Berkeley in 1968 or something, 1968 I think.

Lampson: Something like that. Yeah.

Kay: So you must have met her there.

Lampson: I did, yeah.

Kay: How did you meet her -- because she was a biology wonk?

Lampson: Well, no, she wasn't at the time, actually. She was studying some kind of engineering.

<crew talk>

Kay: So you must have met Lois in your stint at Berkeley.

Lampson: Right. I was a graduate student, she was an undergraduate, and she got a job on the Genie project.

Kay: Oh, she did. I didn't know that.

Lampson: Mel [Pirtle] hired her.

Kay: This is before she had decided for biology?

Lampson: That's right. That's right. She was studying some kind of engineering or mathematics, something like that. I don't remember exactly. And she was working part-time on the Genie Project.

Kay: Then she got her PhD about eight years later, also from Berkeley.

Lampson: Quite a bit later, that's right. That's right.

Kay: But then you had a deal with her that she was going to get to choose the next place you went, right?

Lampson: Well, I don't know if it was exactly a deal, but she got a job at Penn [University of Pennsylvania] and we ended up going to Philadelphia.

Kay: Right. And now, conveniently, she is here.

Lampson: Then she got a job at Harvard, and certainly Boston is a better place for me than Philadelphia.

Kay: One of the questions I was going to ask you was about group dynamics. There have been various kinds of historical writings about ARPA and Xerox PARC, with a wide range of goodness. Some of them have not been very good, and some of them have been fairly good. It's felt to me that the essential thing that has not been covered well enough was how and why people were able to work with each other as well as they did, both in the ARPA community and at Xerox PARC. Have you thought about [that]?

Lampson: Well, there is no doubt in my mind about why it was that way at PARC. It was because of Bob Taylor. Why it was that way in the ARPA community is less clear to me.

Kay: What was special about Taylor?

Lampson: I was not really in the heart of the ARPA community, because when I was at Berkeley I was not a PI [Principal Investigator]. I went to some of the meetings and so forth, but I was not really at the center of it. And subsequently I was pretty much focused on what was going within Xerox.

Kay: So what did you think Taylor did?

Lampson: Damned if I know. It's just magic!

Kay: He was willing to talk about what his plans were, what he thought he was doing. I had a number of conversations with him. He would tell you kind of what he was trying to do. He kind of did it.

Lampson: He definitely had Licklider... I thought you meant: how did he make it work. His goal was perfectly clear. In fact, at the time that we were getting geared up to build Maxc, Taylor tried to talk us out of it, and tried to talk us into building something which later on we could see was the Alto. But he couldn't explain it well enough to convince us that it was feasible. Which it would have been, and we could have built the Alto in 1971 instead of building it in 1973. But we knew that we needed a timesharing system and we did not understand anything about the Alto, and Taylor couldn't explain it. But certainly he had picked up Licklider's vision and was pushing it aggressively. But that doesn't explain why he was able to make it work, because Licklider wasn't able to make it work at MIT, for example.

Kay: Well, I think Lick did what he did, especially the good stuff that he did, kind of instinctively. Whereas Taylor was really a student. He idolized Licklider. and he was kind of a student of what he thought Licklider was doing well. For instance, one of the things he told me in the early days at PARC, was that his goal was to… He used to tell Xerox he would never hire a good person, because you couldn't concatenate good people to do what a great person would do. That was the elitism that Xerox hated from Taylor. But he said his idea was if he could make the social dynamics work, then all of these incredible lone wolves he was attracting would wind up cooperating if it was a good idea. And that they could perfectly well run open loop when cooperation…. It was a very interesting thing. There was a lot of cooperation on hardware projects at PARC, and everybody kind of rolled their own software.

Lampson: For the most part. And that was a deliberate decision.

Kay: Yeah.

Lampson: Well, it was a decision that I argued for very strongly. One or two attempts were made to do things less in that style, and they were pretty unsuccessful. But I think that was primarily because the hardware resources just weren't available. The only way you could make it work, to have a much stronger common software layer, was to bite off the whole giant chunk that the Smalltalk group bit off, and really build a whole different computing world within which to live. And that had a lot of advantages, and they also paid some fairly substantial prices for it.

Kay: But we were able to do it in part because it was laissez faire.

Lampson: Right, right.

Kay: People were happy to let us do it.

Lampson: I don't know to what extent that was Taylor's explicit decision. Of course, what you say can be overstated. There was a lot of groupthink in CSL, and that was not accidental.

Kay: Well, there was Dealer [the weekly group meeting].

Lampson: But more than that, there was a pretty strong culture, and people who did not feel comfortable in that culture typically did not get hired, and did not want to come. That had some drawbacks, because we lost some opportunity to get some really smart people. But I think it was an essential reason for the success of the whole thing, to have that groupthink. You always have to strike a balance, I think. If you want to do that kind of systems work, you have to strike a balance between independence and conformity. The idea that you can just pick people only based on their IQ, or even their IQ plus their engineering skills, and put them together and it will automatically work, I don't think that is true.

Kay: No. I used to hire people by taking my cardboard model of the Dynabook around, and the people who really got stars in their eyes when they saw that, some of them I taught them how to program. Loving the impossible thing was more important than learning…. I knew anybody could learn how to program.

Lampson: For some value of program, yes, right.

Kay: But, of course, in many ways one of the most difficult things in programming is to find end user sensibilities. I think the general programmer personality is somebody who has, above all other things, learned how to cope with an enormous number of disagreeable circumstances.

Lampson: Yeah. And you have to damp that down, to some extent, at least.

Kay: Yeah. It makes it hard to design really higher-level languages with people like that, because they love machinery.

Lampson: Right. Well, this is why PARC did not invent spreadsheets, because we did not have any users for spreadsheets.

Kay: Dick Smallwood came close.

Lampson: Yeah, but not close enough. Not close enough. I used to work at Microsoft for a guy named Nathan Myhrvold, who was kind of a wild man.

Kay: I know.

Lampson: He was responsible for starting Microsoft Research, and for one crazy year in the late 1990s he was co-manager of the Office Group, a job for which he was supremely unsuited. And it only lasted for… Even though his partner was an extremely feet-on-the-ground kind of guy, it didn't work out. But during this year, he wrote a very interesting memo called "The Future of Office." In this memo he said, "From one point of view, it looks pretty hopeless, because depending on how you count, we have somewhere between 2,000 and 4,000 features in this product. And it is getting harder even to think of new features, never mind getting the customers to notice them if we put them in. So maybe the whole thing is a dead end. Progress is impossible. On the other hand", he said, "we should never lose sight of the fact that PC software sucks. And from this point of view there is just a huge amount of room for improvement the customers will value." And, you know, it doesn't suck because Microsoft has a Machiavellian plan to only improve it by 3% a year so people will keep buying the upgrades. It sucks because it is just hard.

Kay: It was really hard.

Lampson: It is unbelievably hard. But there is just an enormous amount of room for improvement.

Kay: Most of this is the kind of the Grand Canyon effect, which is: once you start the erosion going, the river wants to go there.

Butler Lampson: Wants to go that way. That's right, for sure.

Kay: Backing out of the Grand Canyon is not that easy.

Lampson: Well, no, you have to find a way forward. You can't go back.

Kay: Right.

Lampson: That's right.

Kay: Maybe you can divert it in some places.

Lampson: That's a possibility. That's right. But it is very tough. Microsoft made a big effort a few years ago to build a replacement for Office. The reason they did that was that they were plotting a graph, in which time was on X axis, and on the Y axis was percentage of their time that Office developers were using to develop new code. It looked like this [showing a curve going to zero]. They projected forward, and round about 2005 or so it was going to intersect zero. At that point progress would be impossible. And there is precedent for this, right? This happened to OS/360, and IBM had to rewrite it from scratch. So they started this project to replace Office. It's very tough, because obviously the first thing you would say is: I would like it to be dead compatible, because there are all these people that are depending on it.

That turns out to be impossible. Then the question is, if it is not going to be dead compatible, what should it be? Well, that is not an easy question to answer. They ran this project for four or five years. At its peak I think it had about 500 people working on it. It was a complete disaster. Absolute disaster. Because it's very difficult to answer the question: if you get to start with a clean slate and you have given up on complete compatibility, what should you do to meet the needs of knowledge workers? I've got a bunch of ideas, and other people have ideas, but you put them all together in a pot and stir, and you do not get oatmeal.

Kay: One way of looking at ARPA is, because Licklider essentially did not try to make his vision into concrete goals, he was able to fund people who were actually at odds with each other on the vision. Like Engelbart and McCarthy really did not see eye to eye, and there was a fair amount of contention. But the ARPA community, as a whole, worked on a lot of different areas having to do with interactive computing. Somewhat, the degree of stove-piping seemed to help. The fact we had more than one stream of thought going forward at PARC, I think, in the end helped. Because I don't think you can jumble all this stuff together in the beginning.

Lampson: I think that is right.

Kay: Just too much argument, and in one sense…

Lampson: Of course, it is much easier to do that when things aren't so developed. I remember something that Rob Pike, or one of the Inferno guys from Bell Labs, said a few years ago. That looking back on that project, where they developed a system pretty much from scratch starting with the bare metal and working all the way up to applications and users -- I am not sure that I completely believe this, but what he said was that 90% of the manpower went into reimplementing standards, which was boring and not research. But if you didn't do it, you would not have users. That was not a problem that you had in 1975.

Kay: I've heard that argument. That is one of the big web arguments, that regardless of how bad it might have been, if it ever gets embedded in some way, then all of a sudden you've got yet another kind of barrier.

Lampson: That's right. It's something you find, one way or another, you have to live with. The challenge is to figure out how to live with it without completely wrecking things up. The point I was trying to make is it was much easier in the 60s and 70s, because you didn't have all this baggage.

Kay: In 2004 you received the Draper Prize.

Lampson: Well, so did you.

Kay: But this is about you I was going to say, with three other colleagues. But in your remarks when you accepted the Draper Prize, you said a couple of interesting things there. One was just "they give this for people who invented the jet engine", you said. So what were your reactions when you got that phone call from Bill Wolf?

Lampson: Well, I was somewhat surprised because, as I say, they give it for inventing the jet engine. It is a little hard to believe that the Alto is as important as the jet engine, but maybe it is.

Kay: Well, I think of those awards as kind of a crapshoot.

Lampson: Well, of course, there is an element of that, too. Right.

Kay: You are kind of a solution to the problem of seven people in a room over some weekend.

Lampson: But leaving that aspect of it aside, so it was a bit of surprise.

Kay: One of the other things you pointed out -- and this is something that is sort of in line with what we were just talking about -- is you pointed out that essentially nobody else was working on personal computing at the time.

Lampson: That is the part that was always so surprising to us, that nobody else wanted to work on it. As far as I can tell, the basic reason for that was that for whatever reasons, ARPA decided that they did not need to work on it. And they were the source of money, so that was it. But I think if you look at the things that ARPA did fund in the 1970s, it's not that exciting, for the most part. The internet was a big success then. Speech recognition was successful.

Kay: Basically ARPA had become DARPA then, because of the Mansfield Amendment.

Lampson: That's right. That's right.

Kay: Probably PARC wouldn't have happened without the Mansfield Amendment. I don't know whether it would or not, but it certainly…

Lampson: That is possible.

Kay: …one of Taylor's arguments was that this great funding was not going to be the same going forward. So that tilted things in a particular direction. But it was interesting that, for instance, Wes Clark…. I would call the LINC a personal computer, forever.

Lampson: I think it is generally agreed that that was the first personal computer.

Kay: I think it is.

Lampson: Not allowing the TX2 to be a personal computer.

Kay: But aside from that kind of early aesthetic grasp that Wes made, there really wasn't—Pretty much everybody else was trying to make smart terminals of one kind or another. Some of them were addressing… And some of those technologies are still here today. In fact, you could say that because of some of these, what I consider to be real problems with the web, is a certain 3270. [???] is still coming along, isn't he?

Lampson: Oh, absolutely.

Kay: The failure of Java to really be compatible across machines and be the thing it was supposed to be, has led to something that looks an awful lot like IBM mainframe [with] the [IBM] 3270 [terminal].

Lampson: Absolutely.

Kay: The Ajax stuff today, and everything. To me it is frightening. That is why I object strenuously to what I consider to be really artificial, even though there are probably more restrictions to do it generally. I just feel that you should be able to download something safely, even it doesn't talk to anything else except out through the browser.

Lampson: Well, you can do that. Java works pretty well for that.

Kay: Well, but you cannot download Java unless your sysadmin lets you have it.

Lampson: Yeah. But that is pretty widespread.

Kay: Hard to find it in lots of elementary schools.

Lampson: That is because the sysadmins don't allow it, or because nobody thinks it is important?

Kay: I think nobody thinks it is important.

Lampson: O.K. Well, there is that.

Kay: One of the interesting things about the printing press and, I think, the internet and personal computing, is that done right they are basically subversive, even as they are serving all of these purposes. Having little places where the subversions cannot get slid into place without going through some hierarchical thing, seems to be a dangerous thing.

Lampson: That's right. I think that's absolutely right.

Kay: Let's talk a little bit about one of the interesting driving things in your career. Besides writing a lot of theoretical papers and being on a lot of committees, you basically have driven to get pretty much every piece of work that you have done out into the world as a practical thing.

Lampson: Well, that is always the goal, right? For one thing, you never know…

Kay: That isn't a goal for everybody.

Lampson: You never know that it is any good if you have not done that.

Kay: It definitely has been your goal. I subscribe to that goal.

Lampson: Right. It seems like if you can't prove a theorem, then you have to ship something. Those are the only real options.

Kay: And because you have been ahead of the curve on so many things, you have run into all kinds of interesting politics and situations.

Lampson: Yes.

Kay: People who don't understand, and so forth, and various guys. Xerox… well, Berkeley you had mentioned, even. There was Xerox. I am sure you had some adventures at DEC.

Lampson: Oh, heavens yes.

Kay: Adventures at Microsoft, and so forth. What do you think about all that?

Lampson: Well, basically what I think is that sometimes things take time. I don't get too excited about it. These things will find their way out into the world, one way or another. You know what Gordon Moore said years ago: that it's much easier to persuade a venture capitalist to give you money than it is to persuade the management of a large successful company to try something new. That's definitely true.

The management always has good reasons for not wanting to try something new, but it hardly matters whether the reasons are good or bad. They are there.

Kay: But since BCC, you have opted for dealing with the management of a large company.

Lampson: Well, that's true, but that is because if you want to have a research lab, you don't have any choice. But if you ask, "how does the stuff get out there?", it is not usually through that path.

Kay: So that is interesting. I basically follow that path myself.

Lampson: Exactly.

Kay: It just seemed compared to NSF and the difficulty with the DOD funding and stuff, the least of the evils is dealing inside of a company for doing research. But then there are all of these painful…

Lampson: I think I don't get nearly as excited about that as you do. First of all, I think the company is entitled to get value for what they spend, and usually they do. They usually do not get nearly as much value as they could have gotten if they had their heads screwed on straight. But Xerox made a lot of money out of computer printing, which came out of a whole bunch of the technologies that were developed at PARC. DEC made a lot of money out of AltaVista, which was a direct outgrowth of a whole bunch of the things that we did at DEC SRC [Systems Research Center in Palo Alto], even though lots of other cool things that were done in both places were left on the table by the management. But if it gets left on the table by the management, and people leave and they start companies, or they write papers, and other people pick up the ideas, or whatever, so it gets out there. It would be more satisfying if it were more direct. Sometimes it is. Microsoft Word still uses the data structures that were invented by Charles Simonyi and myself in the early 1970s for Bravo.

Kay: Yes. I recognize that. I was hurrying to get this big thing done, and every once in a while I had that interesting thing of the looks on one paragraph finding their way onto another. And I would remember, oh, yeah. This is so familiar.

Lampson: Although that's an interesting challenge, because although that particular… Some aspects of that come out of the data structure, but more than that. I think it is an unsolved problem what the user model for that ought to be. Even after all these years it's an unsolved problem.

Kay: Yeah. I have always thought that the paragraphs should be objects, but then you have all of these interesting problems.

Lampson: That's right. And it's perfectly clear that the users don't think of the paragraphs as objects.

Kay: Absolutely not.

Lampson: In spite of numerous efforts that have been made to persuade that they are.

Kay: No question. I think with the users the structure gets flattened, so objects get flattened into lists, essentially. So you are not in object–land anymore when you're on, because the user basically has permission to violate anything, or else they do not really like the things.

Lampson: That's right. And then the question is what should you do about that? And the answer is, we don't know.

Kay: I think one of the crowning glories of Microsoft Word is the extremely good undo, which is part of its heritage.

Lampson: Yes. You know how that's implemented, by the way. I only learned this three or four years ago talking to one of the Word developers. It is an idea that is so obvious. We didn't think of it at PARC, probably because we didn't have the resources. The data structure that Word uses to represent things is this so-called "piece table", which is basically pointers off into different fragments of text.

Kay: Which indeed is this thing that you and Charles developed at PARC.

Lampson: Exactly. It was one of the two ideas that motivated us to start Bravo in the first place. The way undo works is you hang onto the old piece table, which is not that big compared to the whole document, typically. When you want to undo, you just put it back. Throw the new one away

Kay: Charlie and you invented the piece table in the first place, but not for undo, so it would be a much more compact way of…

Lampson: That's right, exactly. We never thought of using it for undo. And as I say, I don't think it really would have worked on the Alto because there wasn't enough memory.

Kay: The Mac undo was kind of invented by Larry Tessler. It was kind of a semi-object version of that that they created. Whenever you did anything, it was created as kind of a pair, created as a thing.

Lampson: With an undo action.

Kay: With an undo action, which is sometimes just caching something.

Lampson: Right, right. If it is anything other than caching something, it's very hard to make that work. That's right. This technology is essentially the same scheme that database systems use for the undo that they need for transaction abort. You remember the old state, and you just put it back. You don't try to compensate for the actions that were taken.

Kay: One question doesn't quite make sense here, but they wanted me to ask it. The reason it doesn't quite make sense is that you're not really looking back on your career at this point. You are still…

Lampson: Looking forward.

Kay: Looking forward. You're still writing papers. You're still dealing with students. You have a treasure trove of some unpublished stuff on your website that I've been trying to get you to publish. You have this incredible course in…

Lampson: "Principles of Computer Systems", it's called.

Kay: "Principles of Computer Systems". I would love to meet the graduate students who get through this course. So you probably don't spend a lot of time looking back on things. But what do you think when you do?

Lampson: Well, I've had a lot of interesting opportunities, and I've been able to seize quite a few of them. The thing that I tend to get awards for is the Alto, but I did lots of other things, many of which were failures. I give a talk about successes and failures in computer systems research, where there's a fairly long list of things that have failed, like capabilities and functional programming. Usually the audience gets quite upset, because usually there are quite a lot of people in the audience that are working on those things. But I try to temper it by the fact that I worked on about half of them. Remote procedure call, which I worked on fairly hard, has basically been a failure. Security, for the most part, has been a failure. But there are enough opportunities that there can still be lots of successes. I think that is still true. A lot of people seem to think that, well, computing is mature. It is getting kind of boring. There is not that much to do. Well, it's true that because it is big now there are lots of boring aspects of it, but there are still lots of exciting things to do, for numerous reasons including the fact that PC software sucks.

Kay: Yes. Sometimes it's hard to find people who completely would be as vehement about it as you and I are. I guess they have gotten used to the reality.

Lampson: Yeah. Well, you can do lots of good things, but it could be a lot better.

Kay: I just wrote a big thing at the request of some possible funding group about new media on computers that was not simply a digitized version of old media. This is a big idea to them, because their idea is that new media is digitized version of old media, and new new media is being able to link it all together. But the idea that the computer is a special medium of its own, that has some special

characteristics, like the ability to make simulations, that likely could go in a new kind of literacy 50 years from now. You could imagine people arguing with simulations the way people argue with essays today. In order to argue with essays today, the form of argument had to be invented to go along with the technology of printing. We've got this form of simulation, which I think is unique to the computer. It's a different way of thinking about mathematics that we haven't found a form that end users can readily work for that amount of success.

Lampson: Or that anybody can. I have an idea that is related to that, actually, which is I have been scratching my head off and on trying to figure out what one could possibly do about this problem that PC software sucks. Unlike you, I am generally not a believer in new starts, so the question is how can you build on what already exists and make it less sucky. The idea that I came up with, which gets a lot of interesting reactions, is that you should be able to enter into a dialogue with the computer system that you're dealing with. You're working with Word or Excel or something, and then you want to do something, and you can't figure out how. It turns out more than half of the new features that users ask for in a program like Word are already in there, but they just cannot find them. Some people blame the users for this, but that's stupid; the users are what they are. It's the program's fault. But it's very hard to figure out what to do about it. This is why new versions of Word tend to have very annoying features that are in your face, because they know that if they don't turn the feature on and put it in your face, you'll never notice it at all. That's actually, I think in many cases, why they make it fairly hard to turn the feature off even if it's not a very good feature. But with a dialogue, it seems conceivable that the system could understand enough about what it's able to do.

Kay: I'll buy that. Why not? These days you should be able to take an application and make an ontology...

Lampson: Exactly.

Kay: …that covers that application. and the thing should be able to be relatively flexible about those kinds of things. So is anybody doing that?

Lampson: Only in a very sort of toe-in-the-water kind of way. If anyone is making a serious effort to make this work, I don't know about it.

Kay: I just actually got a few bucks from NSF. The thing that I've been avoiding for years, because I have never had a good idea about it, and [was] hoping somebody else would do, is to make a user interface framework that you could put curriculum into, and it will do a halfway decent job of mentoring somebody through. It came to a head on this hundred dollar laptop that I am working on, where the possibility of having the right kind of mentoring support out in the third world, not only in the first and second, is essentially zero. That's a user interface gap that has to be bridged some way or another. It has much of the overlap with exactly what you are talking. Well, the stuff is in there. You should be able to…

Lampson: How can you get to it, right.

Kay: One of the things you should be able to make an expert system about, if you are going to make it about anything, you should be able to make it about a computer system, because it is what it is.

Lampson: It's well defined, for some value of "defined". Yes, right, exactly. There are a lot of interesting angles on it because, of course, any of these rich applications typically has embedded programming, and a lot of the things that people want to do actually are going to require writing programs. And real people are not going to write those programs. The system is going to have to write them itself.

Kay: So we are back. McCarthy wrote a paper in 1958 called "The Advice Taker."

Lampson: Yes. I remember that.

Kay: He pointed out pretty much eventually things will get complicated enough so that the end users will have to deal with the user interface in somewhat common sense terms about things that the user interface could possibly do, including write programs to do various kinds of things. It's kind of an interesting full circle. I would agree with you that that would be… I don't even think of that as much of an incremental step to me, because I think it would be a huge breakthrough if you could do it above some significant threshold for the user.

Lampson: I think it would be, yes. It's incremental in the sense that you wouldn't have to start over. You'd be building on all this stuff that already exists, warts and all.

Kay: I think it would be a breakthrough to have it be above threshold.

Lampson: I think so.

Kay: No question.

Lampson: It seems to me that even if you could do something like this, even if it only worked a third of the time, people would still value it, because the cost to try it out would not be very high.

Kay: Because Help doesn't work a third of the time.

Lampson: Yeah.

Kay: So if you could make something that worked a third of the time, or half of the time, people would <inaudible>.

Lampson: People would really like it, right. And as you pointed out, these systems have pretty good undo, so you can't really get into any serious trouble.

Kay: Basically what you need is instead of an eager paperclip, you need something like a shrugging icon that looked like, "Oh yeah, I don't know. But try this".

Lampson: Yeah. The real problem is that the system has to understand about itself much more deeply than any of the systems do today. The Help really doesn't understand anything. All it has is this fairly crappy search.

Kay: That seems like just like a perfect set of projects for some students.

Lampson: Yeah. It seems like there's a lot of things you could try.

Kay: There is a lot of ancillary technology that has been done now that wasn't there 25 or 30 years ago, that I think you could pull in. So that brings me into asking your thoughts about academia. I used to give a talk where I would say, well, UCLA only has one computer science department but it has seven biology departments. Somebody called me on it at some point and asked me if I had ever counted the biology departments at UCLA. I said, "No, I haven't."

Lampson: Well, does it have seven biology departments? You still haven't counted them.

Kay: No, no, I did. I did go to the…

Lampson: Okay, and?

Kay: First I was going to count the medically related ones in order to make up my seven. But I found, actually without the medical ones, I found 25 full departments with department chairmen, absolutely bona fide departments in biology. What has happened, pretty much when computing started, modern biology started, almost coextensive. But the biologists, every time they get moribund, they splinter off and NIH funds them. Whereas it seems like there's been much more of what a friend of mine used to call the "ingrown toenail" in computing, where there tends to be one computer science department and not a lot of going off. Like there are not big Systems departments. You would think there might be a Systems department.

Lampson: It's very difficult to have a Systems department in a university, because funding is a huge problem and so is staffing.

Kay: Yeah. What do you think about where academic computing is these days? We can promise to excise this from the tape, but I want to hear your opinion.

Lampson: I don't mind goring peoples' oxes. I don't depend on any of these people. I don't think that things are that much different than they've ever been, actually. I think academic computing has always struggled with the fact that it is excruciatingly hard to mount any kind of substantial project in academia. It's excruciatingly hard to get faculty members to work together, because they can typically do ok not working together, and that is the basic dynamic of academia, that you do your own thing. There are domains like astronomy and particle physics where the academics do work together because they don't have any option. But that's what it takes. It has to be that there's no option. I think this is pretty much true in biology, too; they don't work together. It's just that biology has this enormously… Well, first of all people really care about it because they don't want to get sick, or they want to get cured. And secondly, there's just this enormously rich world of phenomena that you can explore, and they're right in front of you. Whereas in computing, you have to make the phenomena. That means that in most cases they're not going to be as interesting, because you have to be pretty good to make interesting phenomena. The biologists don't have that problem, because someone else is taking care of making interesting phenomena.

Kay: And four billion years of making interesting stuff.

Lampson: I think for numerous reasons it's probably not a very good or very useful analogy. As I say, I don't really think academic computing is that different now than it was in the 1970s. In the 1960s it was different, because it didn't really exist, and so it was coming into being. But people have always had the same set of complaints. And, you know, things go up and down. Some things are better when ARPA's funding system is more rational, and so forth. But fundamentally, it seems to me, the academics are good at doing theoretical work, which is what you'd expect. And they're not so good at doing systems work. They tend to do kind of narrowly defined, incremental things where there is a pretty clear criterion for success. Then you can get your paper into an appropriate place in the hierarchy of conferences and so forth. It is very hard to do any substantially innovative thing. I don't think that that's changed.

Kay: MIT, I guess partly because of World War II, developed laboratories as a kind of crosscutting structure. So these laboratories have been able to put together larger aggregates than departments usually seem to be able to do.

Lampson: Well, I don't know. If you compare MIT with Berkeley, which doesn't have this laboratory structure in computing, is it really different? I don't think so.

Kay: Well, maybe not now. Berkeley, I don't think, ever did anything… I wasn't a huge fan of Project Athena, but it was an interesting example.

Lampson: That actually wasn't done by a laboratory at all.

Kay: That wasn't done by LCS?

Lampson: No. No. The stated goal of Athena was to serve the needs of instructional computing.

Kay: Okay.

Lampson: So it wasn't thought of as a research enterprise at all. They did come up with some new things, but that wasn't the framework for it. So I don't think it makes a lot of difference. I mean, when the lab has big chunks of money that it disposes of, that gives you certain kinds of opportunities that might be more difficult to get in other situations. But there's other ways to get big chunks of money, and it's certainly harder for the labs to get them now than it used to be. In fact, at MIT I think what you've seen is that the biggest function of CSAIL [Computer Science and Artificial Intelligence Laboratory] these days is to organize these industrial collaborations. That is something that certainly has some value, but I think other institutions that don't have the laboratory structure have found other ways to achieve the same result. So I don't think it makes that much difference.

Kay: I guess maybe I'm more jaundiced. I agree that it is a lot like it was. It just seems to be more incremental than ever.

Lampson: I think that's true. I think to some extent that's a reflection of the fact that there's a lot more to be incremental from.

Kay: Yes. But in a sense, there are always. That is another interesting….because the medieval world was rich. There is a lot of stuff to be incrementalized from there. I was just reading a book called "The Elizabethan World View", a wonderful little book done by the former head of Jesus College at Oxford, trying to show how the incrementalism of the particular way the Middle Ages looked at the world view was somewhat modified. But it wasn't until science actually happened that there was actually a whole 'nother something that you could call new. And it just struck me that, boy, that is just so much like academia in computing today. There's a lot of stuff being paid attention to, for whatever reason, and it seems very difficult to get onto the next set of stuff that needs to be paid attention to.

Lampson: I think that's true. But I think it's not that different from what it was 10 years ago, or 20 years ago, or even 30 years ago.

Kay: I guess that's why I never went back to academia.

Lampson: Yeah. To a great extent, it comes with the territory, and it's not unique to computing, either. I think you see this in any discipline, that most of what gets done is fairly incremental. Computing is younger, so you have more of a sense of great opportunities.

Kay: Let's look ahead a little bit now. We just talked about something that I think is really an excellent idea that needs to be done, which is this idea of taking things that have been generally found to be useful and have more features than anybody can find, and essentially building something that is almost an entirely new user interface around the thing to make these, and maybe even show how things are done, because these ontologies would have action evidences [ph?] in them, as well.

Lampson: Indeed. That would be critical, because in many cases it wouldn't be that obvious that the system had figured out the right thing. If it can't explain to you what it figured out, in many cases it won't be good enough to just look at the result, because it'll be too spread out. So you'll have to have some explanation of what it is that it is proposing to do.

Kay: You said a few things about interoperability in the future. What do you think about… and this again is one of these tough areas where there is a lot of stuff already operating, and maybe not so well. There are millions of lines of glue code being written.

Lampson: That's for sure.

Kay: What do you think the future of operability could be? Let's take, just as a concrete example, both the existing Microsoft Office and the abortive attempts to do better. One could imagine something where there are not- the overlap between the separate applications in Microsoft Office is large, but also treacherous. One could imagine a better interoperating theory where…

Lampson: Yeah. I guess what I basically believe in is translation. Translation is kind of crufty, but I think in many domains of computing, actually, one has to be very suspicious of the elegant solution. I remember I was having a conversation 20 or 25 years ago with Andrew Burrell about programming parallel applications. Naturally the talk turned to synchronization, and at that time the rendezvous model that was pioneered by Tony Hoare's CSP [Communicating Sequential Processes], and then was adopted by ADA, was very popular. And Andrew said, "There are some things you can do really, really nicely with this rendezvous model." And I said, "Right. That is why I don't like it." Because my view is that there's only a limited amount of elegance to go around, and if you pile it all up in one corner the rest of the floor is going to be extremely bare. Locks are kind of crappy, and they have lots of pitfalls and so forth. But, by God, if you wanted to solve the problem, you can do it with locks. And it will be about equally crappy in one part of the room and in another part of the room. That's maybe not so wonderful, and if you have a big enough domain it's great to have something specialized like APL or Matlab. But I think on a broader scale, you have to be suspicious of that. That's why I like the idea of translation. And I think with more sophisticated systems you can make the translation work a lot better. Yes, it is going to have rough edges, but the problem with the other thing is that unless it's something that you really understand very, very deeply, I think it's going to be very hard to find a paradigm that is going to meet the needs of the diverse applications as well as what they can grow for themselves.

Kay: Take something that is really dog simple, but I think it is interesting in what happens and what doesn't happen with it: just cut and paste. It's one of those things where generally speaking…

Lampson: It is a good example of translation, because in many cases that is exactly how it's implemented.

Kay: And very often the thing gets denatured in a non-recoverable way…

Lampson: That's right.

Kay: …and can't come back again in an interesting fashion.

Lampson: Which is too bad. Microsoft's OLE [Object Linking and Embedding] attempts to alleviate that by having this form of cut and past where you essentially keep the thing in its original form, and all you get pasted, if you like, is the picture. It means it's not integrated with the new world, but it also means that you haven't lost anything, because you can always go back and edit it in the old world that it came from.

Kay: So suppose we were going to do the ontology idea. It seems to me that if a person did the ontology idea, you would actually have kind of a really elevated theory of interoperability, right?

Lampson: Yeah. My prediction is that the paradigm that will win there is the translation paradigm, as opposed to the really deep "it's the same thing in the different applications". That's what we should strive for, but I think we have to be pretty pessimistic about how hard it's going to be to achieve.

Kay: Ok, let me try another one.

Lampson: Whereas with the translation thing, I think especially if you have this notion of ontology and an assistant and so forth, certainly when you map the thing over you can decorate it with annotations that you can use to make it easier to map it back. But it's not as good as having a deep sense in which it's the same in both worlds. I am not opposed to that, but I just think it is very hard, and people tend to grossly underestimate how hard it is. Then what you tend to end up with is something where there was this core of unity, but it grows all these barnacles, and before you know it, you're much worse off than you would have been if you had had two separate things, because the barnacles just made such a hopeless mess of them.

Kay: Let's take your "objects as components" idea, and let's take the Microsoft user interface. An interesting thing you could do is to simply have a switch where you turn off all the window boundaries, and now you can intermix things from different applications on what seems to be the same media space. If you had multiple desktops, then all of a sudden you're making documents now in which it is basically objects, but the objects tend to be large. There is a word processing object…

Lampson: Yes. Well, you can do that today with OLE.

Kay: Yeah, sort of. Sort of.

Lampson: It doesn't really work that well.

Kay: Yeah, but I think it could. If it were more like real objects, but in the separate thing, you might actually have… You know we've been experimenting with those kind of ideas. I pointed out to the X Windows people that if they just didn't show the boundaries around the outside, and just thought about what they were doing already as media, that the desktop already is media, then PowerPoint goes away, for sure, because why copy something over here if you can show it here in its working form. So these are staying kind of with your large… These are not the fine grained object ideas so much as they are saying, o.k., let's go along with Butler and do…

Lampson: Do big objects.

Kay: Yeah. Because the economic arguments that you make for the larger components, more able components, are good. That those are going to tend to be taken care of, and they do come in different types that are fairly recognizable. You can imagine having 15 or 20 different kinds of those things that are really running in separate address spaces. They are really separate dealies. They are not hurting each other or anything else. And they are contributing to the user interface kind of in the way the Mac used to, when you were dealing with one of these things. But by just not having window boundaries around them, you are basically mixing them in a media field.

Lampson: Right. Well, this was exactly the idea behind OLE. A fairly serious attempt was made to make it work. So you can see what worked out. Presumably it could be done better, but you can see some of the things that worked out well, and some of the things that worked out poorly.

Kay: I thought that OLE was actually a pretty good idea. It was just on early underpowered machines also. I think that was a big problem.

Lampson: Yeah. That certainly is a major issue.

Kay: The same publish-and-subscribe stuff that Apple tried that was maybe a little bit too early, but it actually has some of that.

Lampson: But there are some interesting issues. For example, one of the things that OLE does is it tries to make it… The way OLE works, at any given time somebody has got to be in charge and providing the container. Suppose it is Word and you have embedded a spreadsheet, and you want to edit the spreadsheet. There's two modes in which it's possible, in principle anyway, to operate. One mode is you essentially open up the spreadsheet in Excel, and then you're definitely in Excel, and you are messing with the spreadsheet. And when you close it, magically it's there in the Word document. You can even actually see it getting updated in the Word document, but they're separate things. The other mode, which

I don't think actually works -- it's not actually supported when you embed Excel into Word, but it is supported, for example, if you have PDF files embedded in the browser -- is that you actually get the user interface, or some faction of the user interface, of the embedded guy right there as part of the user interface of the container.

Kay: That is, in fact, what we do with Squeak in the browser.

Lampson: My experience has been that generally that has worked out poorly. Because typically the UI's… This is not so much of an issue with the browser, but it's very much of an issue even with a simple application like Acrobat, and it's much more of an issue with an elaborate application like Excel or Word. The UI is so specialized for that set of tasks that trying to mix two of them just does not work well. It just creates an enormous amount of confusion. There's a bunch of things that are not going to work right, and you can't predict what they are going to be, and mumble, mumble, mumble. So, again, I think to really make that work, you would need a much deeper understanding of what the unity is behind these things. That's something that definitely seems worth pursuing, but we learned from the experience with the Office replacement in Microsoft that it's a lot harder than even relatively sophisticated and experienced people think. It seems to me like the kind of thing that really ought to be explored in a research context. When Microsoft was doing it, it was not being done in a research context. They were trying to develop a product. That was clearly a bust. The challenge with doing it in a research context, of course, is that it's a fairly sizable enterprise, and if you do it on too small a scale, you won't actually learn what the problems are, because the big things that you're dealing with are rather intractable. Shortly after I joined Microsoft I learned that there was a recently demised project to figure out whether it would be feasible to modularize Excel. They had written a bunch of tools, and they had analyzed the Excel code base and they had built programs to print out graphs showing what the hypothetical module structure would be and who depended on what, and so forth. After nine months they abandoned this project. The reason they abandoned it was not because you could not see how to modularize the code, it was because during those nine months they observed that the structure that they were inducing had changed substantially. Their conclusion was that unless you could freeze the code for a couple of years you wouldn't be able to make it work, and that seemed impossible.

Kay: One of the things that you said, that you didn't think that the users would learn how to program, which I believe that is true.

Lampson: Certainly they're not going to learn how to program in any of the…

Kay: Any of the known vernaculars.

Lampson: Exactly.

Kay: But clearly, I think, say, 50 years from now, go out far enough, there is going to be something like the expression of intentions from users and the systems are going to find ways of carrying those intentions out.

Lampson: Yeah, absolutely.

Kay: What do you think that is going to be like? In your own research, you've been dealing in a more formal way with intentions. So you're thinking a lot about intentions.

Lampson: Right. But that's the kind of thing that users are not going to be able to handle, it seems to me. Right.

Kay: There is not going to be anything like that. There is no question.

Lampson: It seems to me the closest that we come to this today are things like Mathematica or Excel, where to some extent you can argue that the users are expressing their intentions. The other rubric under which you can lump both of those systems, I think, is the rubric of functional programming. As a result of that, it has the problem that functional programming always has, which is that it works great within some domain and when you get to the boundary, boom, it doesn't work at all.

Kay: Both of them would be pristine for end users.

Lampson: Well, but when you're using Excel you're not having functional programming pushed down your throat. So in that sense…

Kay: You could think of Excel as being retrievable programming. You could think of it more as a huge parallel retriever of things.

Lampson: That's if you hook it up to a database. You could.

Kay: Think of the pattern for seven being three plus four.

Lampson: But that's not the way the users think about it, I don't think.

Kay: Well, basically they're trying to make a value.

Lampson: That's right.

Kay: I think they do think about it that way.

Lampson: Yes. That's right.

Kay: They can find a value. I can combine this. So there's kind of an interesting…

Lampson: But it works up to a point, and then it stops working. So the question is, what are you going to do to deal with that. Jim Morris wrote a wonderful paper around 1980 called "Real Programming in Functional Languages."

Kay: Yes, I remember that.

Lampson: The technical content of this paper was that a language that Jim Horning and I and some other people designed in the mid-1970s, called Euclid, was actually a functional language, which was a big surprise to us, because we never thought of that way at all. The great thing about this paper was the introduction, where he says that many people object to functional programming because they say it's unnatural. That's ridiculous. All functional programming is a discipline, and all disciplines are unnatural. The question to ask about a discipline is not "is it natural", but "is it effective". Then he says that's not all there is to it, because there's different kinds of effectiveness, which we can illustrate with the example of two Japanese disciplines, Haiku and Karate. They have quite a bit in common. They both have been around for centuries. They have an elaborate structure of connoisseurship. You can be qualified in various different ways. But there's an important difference. Haiku requires an appreciative audience, whereas Karate -- if you get into a barroom brawl, Karate will work regardless of whether anyone else in the room knows it or not. Now the question is, is functional programming more like Haiku or more like Karate? The answer in 1980 was definitely Haiku, and I think that's really still the answer, in some sense. So maybe the solution to this is going to be to let functional programming work within its domain and rely on this ontology idea to work at the next level up.

Kay: It has got to be something like that because...

Lampson: But I don't know. Nobody is working on this, as far as I can tell. It's really a puzzle.

Kay: Yeah. I wrote a white paper about, not how to reinvent programming, because I am not sort of desiring to, but I think it would be really interesting to do a model of some substantial part of programming that we've done a number of times over the years that is in a recognizable thing like the personal computing arena, and try and make it a working model. As you say, it's the only way of knowing whether what you are talking has anything there. But try and emphasize the model aspects, and try and make an ontology around the model. That's why I was very interested in what you're saying there, because I think doing that as an intermediate step would be tremendously interesting, and it could actually be a stepping stone to a new kind of program. Because if a system does not have some… The nice thing about the computer, you don't have to deal with the complete common sense world of the human thing that AI gets so tied up in. You just have to deal with

Lampson: The part of it that is reflected in that application. Exactly, exactly.

Kay: And you should be able to nail that down better.

Lampson: I would think so.

Kay: That sounds like a really good thing. I think the last part of this was to get you to do this thing that you probably don't like to do, but you have done it very well a couple of times. You have this wonderful paper called "Hints", which has been widely... Just thinking about the wonderful fact that generally speaking each new generation of graduate students has a lot of potential. To me they are the redeeming thing in a sense. So what would you say, if you are giving a commencement, or let's think of it as an opposite of a commencement speech. This is the actual commencement speech, the commencement of your graduate school career rather than getting the sheepskin. What would you say to people who are thinking about going to graduate school in computing these days?

Lampson: Well, Jim Gray wrote a Turing Award paper on the subject of where is computing going, which was published in the 50th anniversary issue of the Journal of the ACM, January of 2003, I think it was. I thought he actually did a good job of laying out a lot of interesting and exciting problems that, for the most part, people are not working on very seriously, but that seem pretty clearly to be attackable and to be a lot more important than most of the things that people are working on. Another way of getting at it is, I have this model for what computing is all about, which says that fundamentally computing is good for three things that we have thought of so far, anyway. It's good for simulation, and that was the original idea, whether you were simulating nuclear weapons, or ballistic trajectories, or payrolls. It's all fundamentally the same thing, and we rode that. That was pretty much the exclusive thing that computing did for the first 20 or 30 years, and it's still extremely important. And then people figured out that computing is good for communication, and so we got networking and databases and things like that. And I think of storage as an aspect of communication; you communicate from the past to the future. We rode that for 20 or 30 years, starting probably around 1975, and that's still extremely important. Then the third thing is something for which I do not know I have such a good word, but I have been calling it embodiment, which means nontrivial interactions with the physical world, whether it is sensors or robotics or whatever. I think that's going to be the new hot thing, that is just barely getting started. But I strongly believe that that'll be the equivalent of the internet and its importance in the next 20 years. I started saying this a few years ago. I think you can already see some clear signs of this, whether it is something fairly trivial, like the Roomba robot vacuum cleaner, or something not so trivial like the DARPA Grand Challenges for driving cars autonomously. They had one for driving through the desert, which the first year I think nobody got more than seven miles. The second year five people actually finished the 150 mile, or whatever it was, course. And now they are going to have one for driving in the city. So I cooked up this... There was a fad a few years ago for Grand Challenges in computing. I didn't like any of the Grand Challenges that the various committees came up with, because they were boring things like "a teacher for every learner", which seems like that would be a wonderful thing, but it's not very concrete, and you can't really tell whether you have made any progress or not. So I came up with a Grand Challenge, which is reduce highway traffic deaths to zero. It seems like the only way to accomplish that is to make the cars drive themselves, at least in emergencies. That's an example of embodiment, and I am absolutely sure that 20 years from now cars will be driving themselves. Obviously we'll have a lot of other benefits that are, from some perspectives, a lot more important than reducing traffic deaths to zero. You'll be able to read the paper during your commute, or whatever, instead of having to drive. It will be much better. And maybe we could even have a lot fewer cars, because if the cars could drive themselves, then you could just dial one up when you wanted it. And road congestion would get a lot less, because you'd be able to use the roads a lot more effectively, and lots of lots of obvious benefits. It just seems like across the whole domain of things that we do, this is going to have a huge impact. What

things will be like in 50 years, I don't know. In 1997, the ACM had its 50th anniversary, and I didn't go to that, but they had a conference and they invited a bunch of people to say what computing would be like in 50 years. Two things came out of that that I thought were interesting. One was what Nathan Myhrvold said, which was he predicted uploading, which means that you'll be able to upload your consciousness into a computer. Why would you want to do that? Well, because now you can be immortal. That's pretty far out, I think. But a year or two after that, Nick Trefethen, who is a numerical analyst at Oxford, wrote a paper on what numerical analysis will be like in 50 years. He had 10 things on his list, and I can only remember the first two. The first one was, if you have any numerical problem, you will be able to pose it to the machine and state the error bound that you want, and you'll either get an answer that is accurate within that error bound, or you'll get an intelligible explanation of why you can't have that answer. The second one I thought was even more interesting, because he said if you pose the same problem tomorrow, the answer will be different. And why is that? Because the systems will be so adaptive that the fact that they worked on your problem yesterday means that they will have changed their strategy. We'll still be using the error bounds, but the bits will not be the same. So it's hard to know what computing will be like in 50 years. That's a long time.

Kay: Yeah, it is way out there. But one thing that you have said a number of times in talks is that it is very difficult to step into the same river because of Moore's Law. So even people who have had the experience of doing some humungous system and barely surviving through it have trouble the next time around because a lot of the scaling has changed. So they don't really get to do the same system again.

Lampson: Typically you scale your aspirations way up, because you say to yourself, ah, computers are bigger and faster. We can do a lot more.

Kay: So you could say that in the computing field, it's difficult for the engineering to catch up with the learning curve. There is a lot more learning curve on most computing projects than. This is one of the things I try and explain to.

Lampson: Right. It is not like bridges. It is different every time.

Kay: When you were at Xerox PARC where I had a chance to work with you and watch you in action, you led by example. You were able to affect an enormous number of people. I tried to explain this to a later boss a few years ago, that different research leaders have different leverages. You were able to affect at Xerox PARC maybe 40 or more people pretty strongly in a lot of different areas. One of the ways you affected people was by not showing any visible fear about the adventure, even when great risks were being taken and sometimes not everything went according to plan. Having a person around like that really helps the young person who is trying to get into research, because my theory about graduate students is that, in many ways they can think better than we can, but they haven't proven it to themselves yet.

Kay: So self-confidence is a big problem?

Lampson: Self-confidence and where they get it, and how they should think about making mistakes in our field. What would you say to them about- I don't want to put words in your mouth, but if I were going to, I would say that our field is one that is perhaps more, because of all of these learning curve aspects, and because of the youth of the field, we have a lot of mistakes that we need to make when we are trying to push through some of these things.

Lampson: Yeah. I think that's right. But, of course, it's very tough today. Well, it's easier for a graduate student, I guess. But certainly for a young faculty member, it's very tough because taking a lot of risks is not the way to succeed in academia.

Kay: Yeah. It's hard to get tenure.

Lampson: It's hard to get tenure if it doesn't work out. It's hard to get funding, and then if it doesn't work out, you don't really get a second chance, or at least it's not very likely that you will. But for a graduate student, yeah, I think it makes sense to take big risks. It's hard, perhaps, to predict where they are going to come out. There's lots of different ways to get a successful thesis.

Kay: I used to say it was just something you could get three people to sign.

Lampson: That's right. Well, there's a little more to it than that these days, right? Because certainly if you want an academic job, you have to be able to persuade people that you did something cool. But on the other hand, that doesn't have to be that closely related to your thesis. The people that hire you are not going to read your thesis. So, yes, you're not going to, probably in most academic contexts, you're not going to see very many role models for this, certainly not among the faculty. I think that is too bad, but I don't quite see what to do about it. Even if you could wave a magic wand over the funding system, it would still be hard. Once you have gotten tenure, you can take more risks. Look at someone like Martin Rinard, for example. You've heard about failure-oblivious computing? This is Martin's thing that he has been pushing for the last two or three years since he got tenure. The notion is, you are writing a program and it does some bad thing, like dereferences null, or makes a random memory operation, or whatever. Typically what you do under these circumstances is you crash the program. Martin says no, don't do that. Be oblivious to the failure. If it dereferenced null, if it was a write, throw it away. It was a read, give back zero. Plunge ahead! The question is, will the program still do something useful? And the answer is, according to the data that he's collected, yes, most of the time, absolutely. I don't think this would have worked on the 940. My explanation of this is, today's PC is about 30,000 times better than the Alto in every dimension. And it does maybe, what, 30 times as much, to be generous?

Kay: The best benchmark I have ever seen if 50.

Lampson: Fifty, okay. That leaves a factor of roughly a thousand to be accounted for. Where did it go? Well, you know, some of it went into things like internationalization, and interoperability, and this and that. But there is absolutely no doubt that most of the instructions that are being executed are not doing anything useful. So if they do the wrong thing, it doesn't really matter as long as they don't actually cause the thing to go off the rails completely.

Kay: That's true. And, of course, there are a lot of cases where many failures, even at systems levels, are somewhat benign.

Lampson: Absolutely.

Kay: Like SmallTalk does not care about most failures. It just sets them aside and keeps on going.

BLampson: Yeah. This is sort of a variation on a larger scale of Jim Gray's the Heisenbug taxonomy. He says there's two kind of bugs. There are Bohr bugs, which have the property that they are deterministic. If you run the program again, you get the same bug. And then there's Heisenbug. If you run the program again, it will be okay. One of the great things about transaction processing is that you can run the program again. I remember a story in the early 1960s, the MIT computer center was in Building 26, and they had a big glass wall, which is still there. Now there's a big experimental classroom behind it, but in those days there was a [IBM] 7090. And on the opposite side of the corridor there was a room where the consultants sat, and there were two of them. There were two women. One of them was pretty knowledgeable and very nice, and the other one was extremely knowledgeable and not so nice. If you knew what you were doing, you went to Mary Ann, if your problem was not too hard. If your problem was tough, you gritted your teeth and you went to Judy. One day a guy that did not know this comes wandering in, and he storms up to Judy's desk and he dumps down his card deck and his listing. He says, I want a refund." She looks up at him and she says, "Oh? Why?" "This program worked perfectly yesterday and I ran it again today and it didn't work. The computer must have messed up." Judy looks up at him and she says, "You ran the same program?" "Exactly the same." "You didn't change anything?" "I did not change a thing." "Why did you run it again?" So that is the Bohr bug approach, but there is the Heisenbug approach, too.

Kay: That is great. One of the questions they wanted me to ask you...

Lampson: Oh, yeah. Is there really only one question they wanted to ask me that you left until last? Gosh. Because there's all kinds of things we didn't talk about at all.

Kay: What is the most important thing we've left out of this interview? They question they asked me to ask you was what are you most proud of, but the other question is a good one also.

Lampson: Well. We haven't talked much about how the various things I have done fit together or don't.

Kay: Okay, great.

Lampson: Certainly in the PARC world, it was fairly straightforward, because we had this ill formed idea that we were going to have the Office of the Future, and so you needed printing and networking and email and word processing and fonts and databases, and this, that and the other thing. Everything was directed towards that goal. The same thing was true in a somewhat different sense for what you guys

were doing at PARC, where you had a fairly clear goal of building this system that could transform the way that kids used computers, and you brought a lot of different aspects to bear on that. There were some fairly big things there that we didn't talk about at all, like the whole laser printing thing, which isn't that interesting anymore. But certainly the technology was interesting. The details of how we did the technology, there were a lot of cool things that are pretty much obsolete today, because you can blow those problems out of the water with Moore's Law. Later on, when I was working for DEC, it was a lot less clear cut how the things fit together. I worked on a pretty wide variety of different things at DEC. We did a lot of work on programming languages, pretty much following on from the work that was done at PARC. Then we built a multiprocessor workstation.

Kay: That was my favorite.

Lampson: What? The multiprocessor workstation?

Kay: Absolutely.

Lampson: Well, that's now coming into its own, of course, because we have multicore computing, and I would say we have no better idea now than we did in 1985 of what to do with it.

Kay: Our little effort was the Notetaker, which is a multiprocessor computer. That's why I liked Firefly. I think that was a really excellent.

Lampson: Well, it got overtaken by Moore's Law, but now it is back. That's right. And it's a real puzzle to know how we are going make any effective use of all those processes, because it's been clearly determined, demonstrated, I think, that system programmers cannot handle concurrency. So it works well in cases where you can encapsulate a concurrency like SQL. But in general, there doesn't seem to be any good story for that. I think we kind of ran into these problems when we did the Firefly. And we did a lot of interesting work on networking. But it never really came together into any larger whole, I think. I am not quite sure why that is. My guess is that we were just lucky at PARC.

Kay: I think we were very lucky.

Lampson: The environment was just as good when we were working for DEC. Except perhaps at the very end, DEC had a very wide open view of what might be valuable. But there wasn't any overarching goal to shoot for, at least we never found one. I think that's pretty much been true since then, as well. So I don't know what to make of that.

Kay: Well, Licklider's vision was a great vision, the ARPA dream as a vision. I'm certainly very proud to have been part of the PARC experience of trying to round off lots of different parts of that vision. I wish there was an award for a large research. This is what I don't like about awards in computing, because they miss…

Lampson: They're too narrow.

Kay: They miss what's interesting, which is the synergy. Because of the huge engineering components and everything, you just have to have a lot of people who work together. PARC was about as good as it gets in my experience. But I think it was so easily held together by the magnetic [field], just lined up all the different iron filings that were individual people at PARC. We were all kind of pointing at north.

Lampson: Yeah. Well, that was cool, but then…

Kay: Well, gee, that is what we are competing with today, by the way. This is how the whole circle <inaudible>.

Lampson: That's all perfectly true. But then it's an interesting question: why weren't we able to do it again? Because, as I say, I think we did a lot of good things at SRC, but there certainly wasn't any "whole is greater than the sum of its parts" effect.

Kay: I think it probably affected different people a different way, but from my standpoint I was able to put together groups as good as the one I had at PARC, but I never was next to the rest of PARC. The rest of PARC for us was critical, absolutely critical. We needed to have all you super-smart people doing this kind of tolerant interest and disinterest, it made all the difference in the world for what we were able to think about and try to do. So I think there's critical masses and critical masses, and having the right kind of diversity in these. So I think there is a size below you can't go. You concentrate.

Lampson: Well, I think that makes sense.

[audio begins abruptly]

Kay: We were talking about something I think is really important, which is this interaction of goals and critical masses and diversity and all.

Lampson: Right. The goals seem to me to be the thing that's central. For example, we could have certainly found an institutional way to collaborate with you guys in the '80s, if there'd been a motivating goal to make that happen. But there never was. I guess the interesting question is: Was that just a historical accident that isn't going to be repeated? Certainly Microsoft research hasn't succeeded in coming up with anything like that. We work on lots of interesting things, but there definitely isn't any broad vision, even for a 50-person subgroup of MSR.

Kay: My view is that PARC benefited tremendously, because all this had been some part of ARPA. So we already came in as born-again interactive computerists. Taylor was pretty vigilant about not inviting people who--

Lampson: --didn't sign up for that, right.

Kay: --weren't really into that thing. So there was kind of a general leaning forward, and then different expressions of ways of making progress and stuff. It's interesting. The main thing I did after PARC was to do a bunch of things that I couldn't get funding for, and doing the systems stuff. I spent 10 years at Apple studying children in their natural habitats, sometimes on a rather large scale, because the individual differences between children are one of the things that really puzzled me. If you're going to do children's prose, like whether it's Logo or Smalltalk or anything, you always get 5 or 10% who will take to it. But if you're trying to do it as a kind of literacy and want-- like I say, if you don't read for fun, you're not going to be fluent enough to read for a purpose. One of the deals with teaching children how to read is to get them to find the fun in it. Usually different children have different fun with reading. We have centuries of stuff we can find, whether it's the Wizard of Oz stuff or whatever it is that children can find the things that they have fun with and to gradually get more fluent. I think it's really tough in user interface design to come up with a user interface that works for any users, and to have ones that at least don't get in the way of a wide variety or maybe even whisper seductive things in different ways to different users, I think is tough.

Lampson: Well my theory about that actually is it mostly has to do with motivation. If the system does something that you value, then you can adapt to a lot of variations in the user interface. Look at something like WordStar. Tens of millions of people learned to use WordStar, god help us, in the 1980s, even though it had an absolute nightmare user interface from most design perspectives. But the payoff from learning it was great enough that people were willing to do it.

Kay: In a sense, it had the right interface.

Lampson: No, I wouldn't go that far.

Kay: But I mean the totality of it.

Lampson: It delivered enough value, that's right. One of the big problems I think we have with a lot of computing these days is that it doesn't actually deliver that much new value. Therefore, the size of the hurdle for learning to use it has to be a lot lower.

Kay: If nobody had ever seen a bicycle, it would probably fail in most focus groups.

Lampson: Oh, absolutely. Anything new is going to fail focus groups, for the most part, I think. Because you can't really convey to the focus group what the value of the new thing is.

Kay: It is a problem. So that could be part of the ingrown toenail or whatever it is, of consolidating to a kind of a genre. Then the new thing of value actually requires some learning, which of course, the children, if they could see what the value was--

Lampson: --they would cheerfully do it, right.

Kay: --pack together skateboards, or do triple flips on their bikes, or whatever.

Lampson: I think that's the real issue is to find something that has the value.

Kay: Our ten years of stuff is trying to characterize six or seven different kinds of big motivators for children and make sure they are findable in the next system. It turned out it worked quite well. Different children would find different things in there, and that would be the thing that they would want to work on when they weren't in school. For instance, not that I believe in Myers-Briggs, it's kind of an astrological theory of personality, but it's interesting that we don't have user interfaces that can tell how old or young somebody is, or how fluent they are in this, or whether they're interested in this, or any of those kinds of things. That could change what you might show in the surround.

Lampson: Part of the reason for that is that UIs are completely handcrafted still. It's hard enough to get one right, never mind trying to make three or four different ones right, or one that has significant modulations. That probably shouldn't be true, but I'm pretty sure that it still is for the most part.

Kay: I think so. My picture of user interfaces is just hundreds of experiments, most of which fail.

Lampson: That's for sure. Even the most experienced and talented people get it wrong most of the time.

Kay: Yes. But as they say in Hollywood, nobody tries to make a bad movie. But only two out of seven make money.

Lampson: Is that the number? Two out of seven?

Kay: Yes.

Lampson: Okay. That's an interesting number to know.

Kay: A few break even and then the rest--. The head planner at Disney actually famously asked in a meeting, "Why can't we just make the hits?" when he was informed of this.

Lampson: Isn't it you that tells that story about Xerox?

Kay: No.

Lampson: That some senior executive came to PARC and was being showing this and that and we were discussing--whoever it was--

Kay: No, that was different. Let's see. The setting was--

Lampson: I thought it was you.

Kay: No, it was like--

Lampson: You explained to the guy that if you're not failing 80% of the time, then you're not taking enough risks. And the guy said, "Yes, yes I understand that. Just make sure you're one of the 20% that succeeds."

Kay: No, what he said was, "Boy, that's really great, but just make sure it works."

Lampson: Isn't that exactly the same thing as the Disney planner?

Kay: Yes, it is. It is. But it's interesting to hear from an executive in--

Lampson: --in Disney.

Kay: --as a guy who was trying to make plans for this company that was making a few billion of its money from making films and he was wondering why they weren't all making money.

Lampson: Maybe he was the finance guy.

Kay: He was a famous screamer. That's all I remember.

Lampson: Screamer?

Kay: Yes. I think I've covered my list and your list.

Lampson: Well, that's cool. It's gone on for quite a long time.

Kay: Any final words for posterity here?

Lampson: Oh, I don't know. No.

Kay: Thank you, Butler.

Lampson: You bet.

Kay: Thanks very, very much.

Lampson: Yes, thank you.

END OF INTERVIEW

