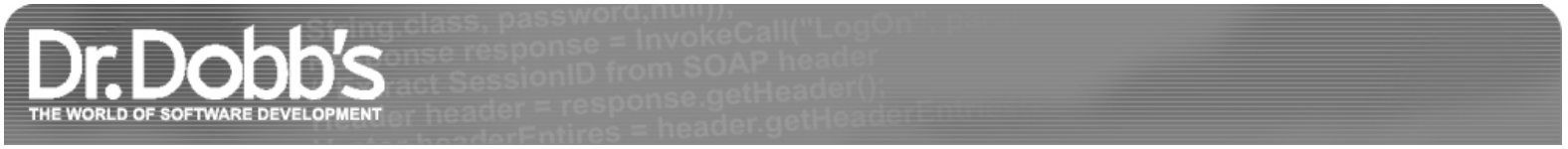

Interview with Alan Kay

The pioneer of object-orientation, co-designer of Smalltalk, and UI luminary opines on programming, browsers, objects, the illusion of patterns, and how Socrates could still make it to heaven.

July 10, 2012
URL:http://www.drdobbs.com/architecture-and-design/interview-with-alan-kay/240003442

In June of this year, the Association of Computing Machinery (ACM) celebrated the centenary of Alan Turing's birth by holding a conference with presentations by more than 30 Turing Award winners. The conference was filled with unusual lectures and panels (videos are available here) both about Turing and present-day computing. During a break in the proceedings, I interviewed Alan Kay — a Turing Award recipient known for many innovations and his articulated belief that the best way to predict the future is to invent it.

[A side note: Re-creating Kay's answers to interview questions was particularly difficult. Rather than the linear explanation in response to an interview question, his answers were more of a cavalcade of topics, tangents, and tales threaded together, sometimes quite loosely — always rich, and frequently punctuated by strong opinions. The text that follows attempts to create somewhat more linearity to the content. — *ALB*]

Childhood As A Prodigy

Binstock: Let me start by asking you about a famous story. It states that you'd read more than 100 books by the time you went to first grade. This reading enabled you to realize that your teachers were frequently lying to you.

Kay: Yes, that story came out in a commemorative essay I was asked to write.

Binstock: So you're sitting there in first grade, and you're realizing that teachers are lying to you. Was that transformative? Did you all of a sudden view the whole world as populated by people who were dishonest?

Kay: Unless you're completely, certifiably insane, or a special kind of narcissist, you regard yourself as normal. So I didn't really think that much of it. I was basically an introverted type, and I was already following my own nose, and it was too late. I was just stubborn when they made me go along.

Binstock: So you called them on the lying.

Kay: Yeah. But the thing that traumatized me occurred a couple years later, when I found an old copy of *Life* magazine that had the Margaret Bourke-White photos from Buchenwald. This was in the 1940s — no TV, living on a farm. That's when I realized that adults were dangerous. Like, really dangerous. I forgot about those pictures for a few years, but I had nightmares. But I had forgotten where the images came from. Seven or eight years later, I started getting memories back in snatches, and I went back and found the magazine. That probably was the turning point that changed my entire attitude toward life. It was responsible for getting me interested in education. My interest in education is unglamorous. I don't have an enormous desire to help children, but I have an enormous desire to create better adults.

The European Invasion In Computer Science

Kay: You should talk to William Newman, since he's here. He was part of the British brain-drain. There was also Christopher Strachey, whom I consider one of the top 10 computer scientists of all time. The British appreciate him. They also had Peter Landin. They had memory management and they had timesharing before we did. Then there was a crisis in the early 1960s. And suddenly the young Brits were coming to the United States.

William was one of the guys who literally wrote the book on computer graphics: *Principles of Interactive Computer Graphics* with Robert Sproull. William came to Harvard and was Ivan Sutherland's graduate student — got his Ph.D. in 1965 or 1966. William followed Ivan out to Utah; then when Xerox PARC was set up, William came to PARC.

A similar thing happened, but I think for different reasons, in France. So one of the things we benefited from is that we got these incredibly well-prepared Brits and French guys reacting to the kind of devil-may-care attitude, and funding like nobody had ever seen before. These guys were huge contributors. For example, the first outline fonts were done by Patrick Baudelaire at PARC, who got his Ph.D. at Utah. The shading on 3D is named Gouraud shading after Henri Gouraud, who was also at Utah — also under Ivan, when Ivan was there.

The Internet was done so well that most people think of it as a natural resource like the Pacific Ocean, rather than something that was man-made. When was the last time a technology with a scale like that was so error-free? The Web, in comparison, is a joke. The Web was done by amateurs.

Computing as Pop Culture

Binstock: You seem fastidious about always giving people credit for their work.

Kay: Well, I'm an old-fashioned guy. And I also happen to believe in history. The lack of interest, the disdain for history is what makes computing not-quite-a-field.

Binstock: You once referred to computing as pop culture.

Kay: It is. Complete pop culture. I'm not against pop culture. Developed music, for instance, needs a pop culture. There's a tendency to over-develop. Brahms and Dvorak needed gypsy music badly by the end of the 19th century. The big problem with our culture is that it's being dominated, because the electronic media we have is so much better suited for transmitting pop-culture content than it is for high-culture content. I consider jazz to be a developed part of high culture. Anything that's been worked on and developed and you [can] go to the next couple levels.

Binstock: One thing about jazz aficionados is that they take deep pleasure in knowing the history of jazz.

Kay: Yes! Classical music is like that, too. But pop culture holds a disdain for history. Pop culture is all about identity and feeling like you're participating. It has nothing to do with cooperation, the past or the future — it's living in the present. I think the same is true of most people who write code for money. They have no idea where [their culture came from] — and the Internet was done so well that most people think of it as a natural resource like the Pacific Ocean, rather than something that was man-made. When was the last time a technology with a scale like that was so error-free? The Web, in comparison, is a joke. The Web was done by amateurs.

The Browser — A Lament

Binstock: Still, you can't argue with the Web's success.

Kay: I think you can.

Binstock: Well, look at Wikipedia — it's a tremendous collaboration.

Kay: It is, but go to the article on Logo, can you write and execute Logo programs? Are there examples? No. The Wikipedia people didn't even imagine that, in spite of the fact that they're on a computer. That's why I never use PowerPoint. PowerPoint is just simulated acetate overhead slides, and to me, that is a kind of a moral crime. That's why I always do, not just dynamic stuff when I give a talk, but I do stuff that I'm interacting with on-the-fly. Because that is what the computer is for. People who don't do that either don't understand that or don't respect it.

The marketing people are not there to teach people, so probably one of the most disastrous interactions with computing was the fact that you could make money selling simulations of old, familiar media, and these apps just swamped most of the ideas of Doug Engelbart, for example. The Web browser, for many, many years, and still, even though it's running on a computer that can do X, Y, and Z, it's now up to about X and 1/2 of Y.

Binstock: How do you mean?

Kay: Go to a blog, go to any Wiki, and find one that's WYSIWYG like Microsoft Word is. Word was done in 1974. HyperCard was 1989. Find me Web pages that are even as good as HyperCard. The Web was done after that, but it was done by people who had no imagination. They were just trying to satisfy an immediate need. There's nothing wrong with that, except that when you have something like the Industrial Revolution squared, you wind up setting de facto standards — in this case, really bad de facto standards. Because what you definitely don't want in a Web browser is any features.

PowerPoint is just simulated acetate overhead slides, and to me, that is a kind of a moral crime.

Binstock: "Any features?"

Kay: Yeah. You want to get those from the objects. You want it to be a mini-operating system, and the people who did the browser mistook it as an application. They flunked Operating Systems 101.

Binstock: How so?

Kay: I mean, look at it: The job of an operating system is to run arbitrary code safely. It's not there to tell you what kind of code you can run. Most operating systems have way too many features. The nice thing about UNIX when it was first done is not just that there were only 20 system commands, but the kernel was only about 1,000 lines of code. This is true of Linux also.

Binstock: Yes.

Kay: One of the ways of looking at it is the reason that WYSIWYG is slowly showing up in the browser is that it's a better way of interacting with the computer than the way they first did it. So of course they're going to reinvent it. I like to say that in the old days, if you reinvented the wheel, you would get your wrist slapped for not reading. But nowadays people are reinventing the flat tire. I'd personally be happy if they reinvented the wheel, because at least we'd be moving forward. If they reinvented what Engelbart, did we'd be way ahead of where we are now.

Objects

Kay: The flaw there is probably the fact that C is early-bound. Because it's not late-bound, because it's not a dynamic system, pretty much the only way you can link in features is to link them in ahead of time. Remember when we had to boot the computer? There's no need for that. There's never been any need for it. Because they did it that way, you wind up with megabytes of features that are essentially bundled together whether you want them or not. And now a thousand system calls, where what you really want is objects that are migrating around the net, and when you need a resource, it comes to you — no operating system. We didn't use an operating system at PARC. We didn't have applications either.

Binstock: So it was just an object loader?

Kay: An object exchanger, really. The user interface's job was to ask objects to show themselves and to composite those views with other ones.

Binstock: You really radicalized the idea of objects by making everything in the system an object.

Kay: No, I didn't. I mean, I made up the term "objects." Since we did objects first, there weren't any objects to radicalize. We started off with that view of objects, which is exactly the same as the view we had of what the Internet had to be, except in software. What happened was retrograde. When C++ came out, they tried to cater to C programmers, and they made a system that was neither fish nor fowl. And that's true of most of the things that are called object-oriented systems today. None of them are object-oriented systems according to my definition. Objects were a radical idea, then they got retrograded.

Binstock: How do you view the Actor model?

Kay: The first Smalltalk was presented at MIT, and Carl Hewitt and his folks, a few months later, wrote the first Actor paper. The difference between the two systems is that the Actor model retained more of what I thought were the good features of the object idea, whereas at PARC, we used Smalltalk to invent personal computing. It was actually a practical programming language as well as being interesting theoretically. I don't think there were too many practical systems done in Actors back then.

Programming

Binstock: Are you still programming?

Kay: I was never a great programmer. That's what got me into making more powerful programming languages. I do two kinds of programming. I do what you could call metaprogramming, and programming as children from the age of 9 to 13 or 14 would do. I spend a lot of time thinking about what children at those developmental levels can actually be powerful at, and what's the tradeoff between…Education is a double-edged sword. You have to start where people are, but if you stay there, you're not educating.

> Extracting patterns from today's programming practices ennobles them in a way they don't deserve

The most disastrous thing about programming — to pick one of the 10 most disastrous things about programming — there's a very popular movement based on pattern languages. When Christopher Alexander first did that in architecture, he was looking at 2,000 years of ways that humans have made themselves comfortable. So there was actually something to it, because he was dealing with a genome that hasn't changed that much. I think he got a few hundred valuable patterns out of it. But the bug in trying to do that in computing is the assumption that we know anything at all about programming. So extracting patterns from today's programming practices ennobles them in a way they don't deserve. It actually gives them more cachet.

The best teacher I had in graduate school spent the whole semester destroying any beliefs we had about computing. He was a real iconoclast. He happened to be a genius, so we took it. At the end of the course, we were free because we didn't believe in anything. We had to learn everything, but then he destroyed it. He wanted us to understand what had been done, but he didn't want us to believe in it.

Binstock: Who was that?

Kay: That was Bob Barton, who was the designer of the Burroughs B5000. He's at the top of my list of people who should have received a Turing Award but didn't. The award is given by the Association for Computing *Machinery* (ACM), so that is ridiculous, but it represents the academic bias and software bias that the ACM has developed. It wasn't always that way. Barton was probably the number-one person who was alive who deserved it. He died last year, so it's not going to happen unless they go to posthumous awards.

> It's like the problem Christian religions have with how to get Socrates into heaven, right? You can't go to heaven unless you're baptized. If anyone deserves to go to heaven, it's Socrates, so this is a huge problem.

Binstock: I don't think they do that.

Kay: They should. It's like the problem Christian religions have with how to get Socrates into heaven, right? You can't go to heaven unless you're baptized. If anyone deserves to go to heaven, it's Socrates, so this is a huge problem. But only the Mormons have solved this — and they did it. They proxy-baptized Socrates.

Binstock: I didn't realize that. One can only imagine how thankful Socrates must be.

Kay: I thought it was pretty clever. It solves a thorny problem that the other churches haven't touched in 2,000 years.

Group Work

Kay: Have you interviewed Vint Cerf?

Binstock: No.

Kay: He's a very special guy. Not just for brains. He's one of the better organizers of people. If you had to point to one person, given that the Internet was a community effort, the one who made that community work was Vint. And he also was the co-guy on TCP/IP. I love him. I've known him for years. He runs a pretty tough, pretty organized meeting, but he does it so well that everyone likes it.

[Digression on who, in addition to Cerf, should have won various computing prizes…]

The prizes aren't a thing that *Dr. Dobb's* worries about, because prizes are mostly for individuals, not for teams that are trying to do serious engineering projects. The dynamics are very different. A lot of people go into computing just because they are uncomfortable with other people. So it is no mean task to put together five different kinds of Asperger's syndrome and get them to cooperate. American business is completely fucked up because it is all about competition. Our world was built for the good from cooperation. That is what they should be teaching.

Binstock: That's one of the few redeeming things about athletics.

Kay: Absolutely! No question. Team sports. It's the closest analogy. Everyone has to play the game, but some people are better at certain aspects.