Stanford eCorner

Finding the Great Problems [Entire Talk]

Barbara Liskov, *MIT CSAIL (Computer Science & Artificial Intelligence Lab)*

22-10-2019

URL: https://ecorner.stanford.edu/?post_type=video&p=64208

Barbara Liskov was already breaking new ground in 1968, when she became one of the first American women to earn a doctorate in the emerging discipline of computer science. After receiving that PhD at Stanford, she went on to design several influential programming languages, including CLU, an important precursor to Java. More recently, as an Institute Professor at MIT and head of the institute's Programming Methodology Group, she has undertaken crucial research on distributed systems, information security and complex system failure issues. She is one of fewer than 100 individuals to receive an A.M. Turing Award from the Association of Computing Machinery. In a conversation with host Ann Miura-Ko, a lecturer in Stanford's Department of Management Science and Engineering and founding partner of the venture capital firm Floodgate, Liskov explores how she discovered the nascent field of computer science, how she recognized and surmounted a number of fundamental computing challenges, and shares her concerns and hopes about how computing will continue to transform our lives.

Transcript

(calm electronic music) - [Presenter] Who you are defines how you build.. (applause) - We have a lot of young people here today and I was fascinated by the fact that you grew up in the Bay Area, you grew up in San Francisco.. Went to UC Berkeley, went away for a little bit.. Came back to Stanford, so we'd love to hear a little bit of the origin story.. - Okay, well as you said, I grew up right in San Francisco.. It's a very different place now then it was then.. And I was always interested in math and science.. Even though at that time, even more so then today, it was not considered the thing that women should be doing.. But I persevered, and I went to Berkeley, and I majored in math.. - Were there early influences that pushed you on that path? - You know, I don't think so..

I think that my parents, my mother was a housewife, my father was a lawyer.. They had no particular reason to push me toward engineering and so forth.. But they were certainly encouraging.. Whatever I wanted to do was fine with them, which I think is really important for young people.. So I went off to Berkeley, I majored in math.. And when I finished there I thought about going to graduate school, but I decided I really wasn't ready because I didn't wanna make that kind of commitment to studying.. And so I decided to get a job.. And I had a close friend who had graduated from Stanford, and we decided to move to the Boston area because my father grew up here.. And I came to Boston, and I went to look for a job.. And I couldn't find a good job as a mathematician..

Which is not surprising, since you really need more than an undergraduate degree in math to do interesting math.. But I was offered a job as a programmer.. And at this point I didn't even know computers existed.. I had never heard of a computer.. And they were hiring people like me who knew nothing about computers, because there was no computer science department.. There was nobody graduating from college who knew how to program.. So they would hire anybody that they thought might be able to do the job.. And I went to work at a place called the Mitre Corporation.. And my first day on the job I was handed a Fortran manual and I was asked to write a program to do some little thing that they described to me.. I've forgotten what it was..

And I discovered this field that I knew nothing about.. That was really interesting and really fit my abilities.. And so I was off to the races, and that's how I discovered computer science.. I worked at Mitre for a year and then I switched and worked at Harvard for a year.. At Harvard, I worked on a machine translation project.. And I was actually maintaining a big program.. In those days you used to have printouts, and that printout was two or three inches thick.. And my job was to maintain that code.. And it was written in machine language.. So one of the two benefits of that job, I learned how the machine works, so I learned about computer architecture..

And I also learned about what it means to write good and bad code, because when you're maintaining somebody else's code, you certainly see both the good points and the bad points.. Anyway part way through that period, and working at this

job, I decided to go to graduate school.. Because I felt that although I was learning a lot, I was basically self-taught.. And it seemed like I could make a lot more progress going to school.. And I decided to go back to Stanford because I felt it was time to come back to California.. So that's sort of how I got into the field.. - And when did you come back to Palo Alto then? - I came here in 1963, so I worked for two years and then I went back to graduate school.. - A lot of undergraduate students are thinking about this, in this moment right now, of do I go straight into graduate school, or do I take a moment to go out and work for a little bit, and then come back.. It sounds like this process of actually getting out and working for a little bit, was very influential in what you decided to study.. Can you talk a little bit more about that? - I think it was obviously crucial for me..

If I had gone on, the way I was going when I finished my bachelors degree.. I probably would never have ended up where I ended up.. I think that in a way, what happened, because I went and looked for a job, is that a door opened, that I hadn't expected to see.. And actually, I think these kinds of moments happen in many people's careers.. You think you're on a certain path, you discover along the way, some opportunity presents itself, and you move in a different direction.. It just changes the way your life is.. As far as the question of, should you go on for a graduate degree, right away, or is it okay to go and work instead.. I always tell my students, that it's a personal decision.. What you're looking for in a career, is something that you're good at, and something that you like.. Because if you don't have both of those things, you're not gonna be successful..

But exactly what that's gonna be for you, depends on you, and you have to find a path and figure it out for yourself.. And so there isn't a right or a wrong answer here.. It's just, you sort of follow your nose and things will happen.. - I love that.. So you arrived here in Palo Alto in 1963, for many of you in the room, you all know that this is, Palo Alto is ground zero for many very well known startups.. In 1965, there was a very well known startup, that started in Palo Alto, called the Grateful Dead.. On High Street, and so that's the period you're coming here to Stanford.. So I would love to know what was Stanford like in that period, and then what was the computer science department like.. - So Stanford was a backwater.. I had been going to Berkeley, Berkeley is so full of energy, there's so much going on there..

- Still is.. - And I went to Stanford and it just felt like I was in the middle of nowhere.. And you have to understand that it was much less developed then it is now.. There were still apricot orchards in this area.. So it was a different time.. - But they had the Grateful Dead.. - They did have the Grateful Dead.. And the whole business of startups wasn't really around, not the way we know it today.. HP existed, so there were some more electrical engineering things.. SRI existed I think, but there wasn't a hell of a lot going on..

But as far as a place to study was concerned, it was good.. There wasn't even a computer science department yet, when I got here, it was just a program.. It was very small.. I don't actually remember how many people were in my entering class, but it was probably five, or something like that, really small.. Nevertheless, the classes that I took had quite a few people attending them.. And I remember fondly, the class I took on compilers, in which the class took over the machine room in the evening, because it was the only way where we could get our turnaround in time.. Since that was the day of batch processing.. Otherwise you would submit a deck, and two days later you might get back the answer about whether there was a bug in your program.. - And you were the first woman to get a PhD in computer science from Stanford.. - Yes..

- Were there challenges there, or did it feel different, or were you used to being a woman, a pioneering woman in your field, by this-- - (laughs) Well I was used to being one of one or two women.. In Berkeley, when I was taking my math courses, I was usually one of two women in those classes.. So that part did not feel unusual.. I actually found Stanford very friendly.. I mean, I felt the students were helpful to one another, when I talked about what was going on at the compiler course, that was all of us together.. So I found it a very supportive environment.. - Fantastic.. How did you decide what you wanted study, once you were in this computer science program? - My recollection, which I am completely certain must be totally false, is that on the day I arrived, I met John McCarthy, walking up the steps to the computer science building, and I asked him whether I could be an RA.. Now I very much doubt that this is actually true, because I don't think I would have had the courage to have asked that question.. On the other hand, I also think that they expected me to go into AI, because I had been working at Harvard on this machine translation program..

Although I wasn't doing any of the technical work.. I was just maintaining the program.. At any rate, I started working with John McCarthy, and I did my thesis in AI.. I came to realize, part way through, that I really didn't want to continue in AI.. - Can you tell us a little bit, what was AI back then? - So AI back then, was different.. Then, the idea in AI was that you would design programs that worked the way people did.. So I was particularly interested in machine learning, but the idea was that the machine would somehow learn, just like a person would learn.. And that was a really hard way to do things.. And it wasn't terribly effective.. It's totally different now..

What happened with machine learning, and especially deep learning, is it changed all of AI and so things are really quite different in AI now.. And what's going on today, is totally different then what was happening then.. - Interesting, let's turn to your Turing Award citation, which reads, "For contributions to practical "and theoretical foundations of programming language "and system design, especially related to data abstraction, "fault tolerance, and distributed computing".. - So then I'm gonna tell you the story of what happened as I finished up at Stanford.. So I had decided that wanted to move out of the field of AI.. I also decided that I wanted to do that, after I finished my PhD, because I thought it would be faster to finish, and after all, the point of being in graduate school, and getting your PhD, is to finish up and go one and do something with it.. So I

finished my PhD in AI, it was on a program to play chess end-games.. And in those days, chess was a sort of a killer app for AI.. The reason was that computers were not very powerful.. And they just could not do the kind of search that's needed to figure out which move to make..

And so that was why I was working in that area for my PhD topic.. Anyway, I finished my PhD, and nobody was willing to offer me a good job as a faculty member.. And so I ended up going back to Boston and working for the same company that I had worked for originally, Mitre Corporation.. It was all my husband's fault.. Because I wanted to move back to Boston because he lived there.. So I got to Mitre, and it was actually very fortunate that I did not go on to a faculty position right away.. Because it's not easy to start up as a new member of the faculty, and at the same time, and you're teaching courses, and you've all these obligations and you're also trying to change fields.. And I was making a major change of field, from AI into computer systems.. And being at Mitre, gave me the freedom to do this.. I was in a research position now..

And the first project I worked on, was a time-sharing system, which time-sharing was a hot topic at the time.. And I worked on that for a couple of years.. And then I was at Mitre, and they do research for the government, and I was asked to look into this problem that the government was interested in.. Namely, what to do about the software crises, so the software crises was, people would build big programs, and they wouldn't work.. They spent, millions of dollars, hundreds of man years, and in the end they'd have to scrap the whole thing.. And actually, in the '60s, the '70s, the '80s, you could read in the newspaper about these fiascos.. Company such and such spent all this money and now they've had to throw the whole thing away.. So the software crises was a really big problem.. And I was asked to start thinking about this.. And so I started to look into this field, it's called programming methodology..

Of course I read all the papers that existed.. And there were some really good people working in that field, there was Edsger Dijkstra, Tony Hoare, Dave Parnas, I mean these were very good people, and they were writing papers about, how do you break up a program into pieces so that you can reason about it.. The problem that they were worried about, was software programs are huge.. They were huge then.. Millions of lines of code, they're even bigger today.. There's no way that you can make sense of something that big, you have to have a way of breaking it up into small pieces that you can work on independently.. Reason about independently, and then somehow you put the whole thing together, and it works.. And nobody knew how to do that.. So they were talking about things they called modules.. But Parnas said, I don't know what they are..

There's these things called modules, but I don't know how to describe them.. And they were also worrying about how do you do design.. And Niklaus Wirth wrote a paper about top-down design, and he sort of talked about it, but it was unrelated to the software structure that existed underneath.. Anyway, I read all these papers, and I realized that I had invented a software methodology, when I was working on my first project, the Venus system, because that was a complicated project at the time, and I was very worried about how my small team of programmers would manage to build all that software, and have it work in a short period of time.. And so what I did was, I sort of broke the rules and the way that programs were being built at the time.. In those days, there tended to be lots of global variables.. And then lots of code, and the code interacted through the global variables.. And that didn't actually work all that well.. So what I decided to do, was to say that I was gonna not have any global variables, I was gonna partition them into discrete units.. And there would be some code responsible for each partition and the only way that code could interact with one another, and that you can get access to the globals, was by calling operations that the partition that was in charge of those globals, provided for you..

So I had this notion of what I called, a partition, or a multi-operation module, that had some data hidden inside, and a bunch of operations that you would use to interact with that.. And I wrote this up.. - Which is sort of a fundamental concept you encounter in a CS 106A class.. - Well we hadn't quite got there yet though, so this was at Mitre, so here I am at Mitre, and meanwhile I had written a paper on my operating system, Venus.. And I submitted it to SOSP, which is the top conference in systems.. And I presented it at this conference and unknown to me, there were people from MIT, kinda looking for women.. So what had happened, is Title IX was on the verge of being passed.. And Title IX, although it has to do with athletics, actually started to open the door for women.. And the president of MIT, Jerry Wiesner, I think, was interested in having them hire some women.. And the electrical engineering department, which was all that existed at that point, had gotten the message..

And so they were kinda looking, and the chair of my session, was a professor at MIT, and there were a couple of other senior professors in the audience, and as a result of this talk, they asked me to apply for a faculty position.. And so I moved to MIT in the fall of 1972.. And this was actually a really good time to make this move, for me, because at that point, I was totally wrapped up in the program methodology question, and particular, I wanted to understand what can we do to help people figure out how to break their problem.. When they're doing design, into a bunch of modules that make sense.. And nobody knew how to do that.. And the benefit of being a professor, was that I could do my own, I could define my own research direction.. And I had a research direction that I was really interested in pursuing.. And this was consuming me, sort of night and day.. I was thinking about this all the time.. This is during my first year at MIT..

So I'm teaching a course, and I'm thinking about this research.. And the way I have always done my research, is to focus very hard during the day, and then I stop work and I go home in the evening and I don't work.. But the thought I was doing during the day, it's in my subconscious.. And so I'm still thinking about things, even if I'm not thinking about them.. And a common situation for me, would be that when I'm driving into work in the morning, and I'm kind of thinking about what am I gonna do today.. All of a sudden I see a solution to a problem I hadn't been able to solve, the night before, the day before.. So

the subconscious, the power of the subconscious and also making sure you're not too tired, is a very effective way of doing your work.. So I'm thinking about this question about how to do modularity, and all of a sudden I had this insight.. I thought of data abstraction.. So before, I had talked about these modules with multiple operations..

But they were disconnected from any notion that might make sense for design.. And when I saw that this could be connected to the notion of an abstract data type.. That made a huge difference.. Because people already understood about procedures as a way of breaking things into modules.. Those are abstract things, like a sort routine, which is sort of independent of how you implement it.. But it's an abstract idea.. Same idea here.. I could have a set, I could have a stack, I could have a tree, whatever it is, I could think about that as an abstract concept.. I could invent it during design, and then later, I could figure out how to implement it.. So it gave me a way, not only to break things up into much bigger, more effective modules than what we had before..

But it also connected to a design process, where I could think about things abstractly.. And then focus on how to implement them, as a second step.. So that was this wonderful idea I had, in 1972 some time, or 1973, which was the invention of the abstract data type.. - Fantastic, and that's sort of a foundational element that we see, that influences how we teach computer science today, at a very basic level.. - Yes.. - So taking us to a more modern era, where I'd love to cover one of your other areas of study, which was in distributed computing.. And you worked on consensus algorithms, particularly with your grad student, Miguel Castro.. And I was particularly drawn to this paper around Practical Byzantine Fault Tolerance, because it's been very influential in a very new area of cryptocurrency, and so, before we ask you about the research, I think it's important to know if you own any cryptocurrency? - (laughs) I absolutely do not own cryptocurrency.. I feel like one of the things that happened as we became more and more civilized, is that we learned how to control currencies.. So that they're less of a gambling mechanism, and more of a, you know, they have some value associated with them..

I'm not very interested in cryptocurrencies.. - Some people would say bitcoin has some actual value, but we can disagree with them.. - It fluctuates wildly.. (laughs) - It does, it does.. So I'd love to understand more around how you became interested in consensus algorithms and for the laypeople in the room, what is this? - Okay, so we've jumped way ahead, so I'm gonna continue my story.. - Okay, - Okay, so, (laughing) after I invented data abstraction, I then spent many years designing a programing language called CLU, which was the first language that could support a data abstraction, as a built in concept in the language.. And ultimately, CLU, together with the work of Alan Kay, on Smalltalk, and work that followed on from that, is what object-oriented programming is all about.. And so then Java, when it came along, picked this up and put it into the mainstream.. Meanwhile around about 1978 or so, when I finished the work on CLU, I was looking around for the next research project.. And I happened to read a paper by Bob Kahn, who is one of, along with Vint Cerf, is one of the people who invented the internet..

And Bob and this paper, talked about his dream about distributed computing.. The idea was that I would have programs that would have components running on many computers, connected by a network.. And this seemed like a really interesting idea.. But nobody knew how to do it, so I thought, gee, there's a really interesting research topic, and so I decided to jump into distributed computing.. So I switched the main focus of my group, in the 1980s, to distributed systems.. And one of the things that was interesting in working in distributed systems, was that remote file systems came into existence at that time.. And I was a little disturbed about remote file systems, because in the old days, when you had your files on your computer, if your computer was up, your files were accessible.. On the other hand, once you had a remote file system, there were two reasons why you might not get to your files.. Your computer might be down, but maybe your computer is working just fine, but you can't either access that system, 'cause the Internet's down, or that computer has failed.. And this seemed undesirable..

And so I started to think in the '80s, about how to solve that problem, and I worked with the student whose name was Brian Oki, on a precursor to the work you're talking about.. Which was, how to build replication systems.. The way you solve this, is by having many copies of the data.. But of course you have to manage that carefully because if there are failures of concurrency, you wanna make sure that the answers you get tomorrow, are the same as what you think you saw today.. So you don't lose data, doesn't get confused, and so forth.. So Brian and I worked on a replication algorithm that handled what are called, benign failures.. Or crash failures.. The idea was, your computer's either up and working, or it's completely silent, you don't have to worry about it.. And similarly for network, the messages are delivered, or if they're bad, they're recognizably bad, or everything's, or they're completely silent, again.. So a simpler failure model then what we have today..

And with Brian, I designed a replication technique that worked in that world.. And that was in the '80s.. And then, in the '90s, I had this very talented student in my group named Miguel Castro, and Miguel was looking for a thesis topic.. And I said to him, why don't you go out and look for the Requests for Proposals, the RFP's that DARPA has put out, and see if there's something that strikes your fancy, out there.. And by then, now we were reaching the Information Age.. Where lots of people had access to computers.. They had computers in their homes, they were connecting, and so forth.. And we could already see all the bad behavior that we see so much of today.. So people hacking into computers, reading your messages, changing your messages.. Faking your messages, all that sort of stuff was coming to the surface..

And so DARPA, which is the defense agency research arm, was asking for people to look for things that might help with this problem.. And Miguel thought, why don't we work on Byzantine-Fault-Tolerant Replication.. So I think this came from two

places.. I think partly, in my group, we were maybe the only group in the country, that actually understood how benign fault tolerance worked.. Because this stuff wasn't really well understood throughout the computer science community.. But the people in my group, they all knew it, because we had developed this way of solving the problem.. So that was in his mind.. And that enabled him to think about how can we take this next step, to this harder thing.. So, here's what a Byzantine failure is; instead of having the computer up or down, the computer's up, but it is acting badly.. And it might even be acting badly in a way you can't detect..

So you might say it, store my file, and it comes back and says, file stored.. But in fact, it threw your file away.. Or it corrupted it, or something like that.. So it misbehaved.. And similarly, the network is malicious.. Now we have all the kinds of problems we know exist, on the network, where people fake their messages and they corrupt your messages and they steal your messages and so forth and so on.. So you have to worry about these attacks, coming in both fronts.. And so we started to work on this problem.. And because we knew about how to do the other kind of fault tolerance, that gave us a big leg up.. Because we could think about this as something we could build on top of..

We could extend that way of doing things to handle this much more difficult system.. And so Miguel and I worked on that.. We started maybe around 1996, we finished that work around 2000.. And that's the system you're talking about, called, Practical Byzantine Fault Tolerance.. The reason the word "practical" is in there, is because my group has always been an engineering group.. We want to make things work in a way that you could really use in practice.. There was work already, in theoretical Byzantine Fault Tolerance, but those were not algorithms that you would really wanna use, if you were really building a system.. So we were looking for something practical.. And so here it is, about 2000, and Miguel and I had finished this work.. And we were asking ourselves the question, will anybody ever use this stuff? That was one question..

And also, we wondered about how many replicas you had to have, in order to do Byzantine Fault Tolerance, if you wanna survive F failures, so F bad computers, you have to have 3F plus one, replicas.. So it's a big multiplication factor.. So we thought, well, maybe F equal two, F equal three, that was about as much as we thought, could be stood.. And we thought maybe this might be used for a key distribution center.. Which is where the keys, the public, private keys that people use on the internet, come from.. One thing I should mention, is around 2000, finally, practical, the simple benign failure stuff that we'd done, started to appear in products.. So that was a big lag factor of about 12 years from the time that work had been done, till the time it started to appear.. So even though we couldn't see where this would go, we might have felt hopeful that it would go somewhere ten years down the road.. So that's where we are, we stopped working on this problem.. Miguel and I, Miguel went off to work for Microsoft Research, in Cambridge, England..

I continued in my work at MIT.. And then, I think it was 2008, is that right? The bitcoin paper came out.. And that is the killer app for Byzantine Fault Tolerance.. The result has been lots and lots research has been building on the stuff that Miguel and I did, back in the late 1990s.. - So everyone, Satoshi Nakamoto right here.. (laughing) - He had a different technique mind you.. - Well I wanted to switch directions a little bit, if that's okay.. - Sure.. - I know that you have actually, you have a son who also is a computer scientist.. - Right..

- I think it's also wonderful that you've written a paper together? - Um hmm.. - So two kind of general questions here.. A, the path to getting to a point where you get to write a paper with your son, what's that like? But secondly, so as a mother, as you think about raising children today, in this computer science era, how should people think about that? - (laughs) Let me take the first one first.. - It's the easier one.. - And then remind me about the second one.. I feel that in a way, my husband is responsible for the way my son came out.. Because whenever I was out of town on business, which happened from time to time.. They would go to Friendlys, which no longer exists, but it was a chain of restaurants.. And they would sit at the counter and have their food, and work on math problems.. And my son is a much better mathematician then I ever was..

And so he went on to, he went to Harvard as an undergraduate.. He started off in math, he actually ended up in computer science.. But he's always been more theoretical than I am.. And so he ended up in theoretical computer science.. When we did that work together, this was on Secret Sharing, which is another crypto kind of thing.. And he was sort of the crypto, the theoretical side of this, and I was the more practical side of this.. I'm not really responsible for the fact that he ...(laughs) I think my husband gets the credit for that.. But I can tell you that it is a fabulous experience to be able to do this kind of work with a child.. I mean, it's wonderful.. - And then, as we think about how tech is now so integrated into our world..

And for you to have seen, that technology's become increasingly pervasive in our lives, I'm curious to know, what are you most optimistic about, in the world, and then, what are you worried about? And you can take either direction.. - Well I'll start with the good stuff.. - Okay.. - Okay, so it's clear that computers are doing a lot of good stuff for us.. And one thing that comes to mind right away is elderly people in their homes who can now have sensors that will detect if there's problems.. Even being able to use remote medicine.. I'm not talking about AI now, I'm talking about just being able to talk to somebody, or pick up advice and so forth.. You think of any field, and you can find ways that computers are helping.. Medicine is an easy one to think of, but there are many of them.. So you can see many of these benefits coming along..

Unfortunately you can see many problems.. And the problems are what worry me more.. I think about them, more than I think about the benefits.. And I worry about things, like fake news, I worry about job displacement.. I worry about the spread of bad stuff on the internet.. I worry about the compartmentalization of communities that sort of reinforce one another's

views.. There are just lots and lots of problems.. Now in the days when this technology was being developed.. I think, to some extent, people were very naive.. But maybe they were also a little blind..

Blinder then they should have been.. Because I can remember people saying, oh, we shouldn't have these people that are developing chat rooms, be at all responsible for the content that people are putting into their chat rooms.. Because that would slow down development, and get in their way.. But you know, if we'd thought about it then, I think we could have seen what was coming.. And I worry about that.. And I think that we're facing a very difficult time now.. It's always difficult when you have a displacing technology, which I think is the Information Age is a really displacing technology.. It takes time for people to adjust and learn how to live with that technology.. Some of the worries about job displacement are probably, maybe not an issue, because always, in the past, other jobs have come along to make up for the jobs that are missing.. I don't know..

But I worry about the ethical issues.. This whole issue about privacy, versus free speech, versus security, all of that stuff.. And I think some of these problems have technology that can help, but they're bigger than technology, it's gonna require changes to laws, changes to the way we think about things.. So I worry about that stuff.. - Does it change then, the way we ought to educate our engineers and computer scientists? - I think that we should be teaching ethics to our students.. I think we should, bringing up these questions, I think we should tell people who are doing innovative work, especially in Artificial Intelligence, people are working in security, that's been an issue for a long time.. But I think people who develop the AI, should be thinking about these problems.. And thinking about ways to counteract them, and their systems, to the extent that that's possible.. So yeah, I think we should be teaching this stuff.. - Fantastic..

With that, I would like to open up questions to the audience.. - [Audience Member] Okay, I have a question that's on AI.. What do you think about the bias that is associated with the AI, and how we, as engineers, we can actually eliminate that bias? - If you wouldn't mind standing up and talking a little bit louder, that would help.. - [Audience Member] Okay, what do you think about the bias that is related to AI systems, and how do you think, as engineers and as people as a whole, we can actually take action to eliminate this bias, to make-- - Okay, so I think this is a question about bias, and I believe you're talking about stuff like the extremely naive statements that were made, when they started to apply machine learning to hiring? And they would say, oh, this is gonna get rid of all the bias in the hiring.. Which is totally false.. Because of course, the training data reflects the bias of the people who made the decisions previously, and so it goes right into the algorithms.. The way you counteract this is, I think, to be aware of it, and make sure that you are counteracting the bias in the training data.. So that this will be reflected in the decision-making.. I think you probably also have to keep watching out for the decisions that the algorithms are making.. And checking them against what kinds of decisions are these, and are they the right kind of decisions..

It's actually interesting, at MIT, I worked for a while as an Associate Provost, where I was in charge of diversity on the faculty.. And what's really important, when you're in a position like that, is to collect the data, and analyze the data.. So you wanna know what was the applicant pool? How did this pool get reflected in the people who were interviewed? How did the interviews get reflected in the people who were hired? How did that all work out? So you want a lot of that kind of analysis going on as well.. So that's what I think you can do.. Exactly how you keep the bias out of the training data.. That's an interesting question.. And it's another point, by the way, as I understand it, people don't really understand how those learning algorithms work.. So as they come to understand them better and better, I think this will also help with that problem.. But I'm glad you brought that up.. I mean, that's an early example of how people were just dead wrong about what that algorithm was gonna accomplish..

- In the very back there.. - [Man In Audience] Can you hear me? - We can't hear you, say it loudly.. - [Man In Audience] This is like an amazing perspective that you're providing, I haven't even thought about object-oriented programming being invented, and so my question is, you've seen the growth in our capability of computer science over the last 50 years.. Does that make you optimistic, or less optimistic about how far computers should be able to go? Versus what humans, for example, are able to do.. 'Cause you've seen it from nothing to where it is now.. And so you have-- - Wasn't that far that she's-- (drowned out by laughter) - You mean it wasn't the stone age.. - [Man In Audience] Yeah, does that make you more or less optimistic about the limits of computer science? - So, as I understand your question, you are thinking about this, sort of the ultimate AI question, of, can a computer be as good as a person.. Or mimic a person.. I don't know how to answer that question.. I don't think so..

But maybe that's just wishful thinking.. (laughs) I don't see computers having the creativity that people have.. I don't know.. I haven't got a crystal ball here.. But I sincerely hope that in fact, computers will not be able to be people.. - Think we all hope that.. Yes.. - [Person In Audience] My question is, can you think of a time in your career, where someone you deeply admired or respected, advised you to do something, but you decided to, or chose to do something totally different.. How did you come to that decision, and how did you feel at that time? - So this question is sort of related, the question is, did I ever have somebody who advised me to do one thing, and I decided to disregard the advice and do another.. I think in some sense, my whole life has been like this..

(audience laughs) So if you think about, here I am in high school.. And girls are not supposed to be interested in math, but I didn't let that stop me from taking all the math and science courses I could, and there's just been a kind of a continuing story here.. And I've always kind of wondered why it was that I was able to do that, when a lot of women were not able to do

that.. And I don't have the answer to that.. I have a sense of humor, I often think stuff is funny.. When other people might get put off by it.. But I think I may have been lucky as well, I never had any of the horrible things that sometimes women report, happen to me.. I don't know why.. So, in answer to your question though, there's another thing I wanted to say, which is when it comes to advising faculty, I feel it's extremely important, and I think this must apply all over place.. But I'm thinking here in particular, at a university, we have junior faculty, or trying to get tenure..

And there are senior people trying to look out for them.. I think it's a big mistake for senior people to start telling junior people what to do.. Because in the end, you have to be responsible for yourself, and if somebody tells you to do something, and you do it, and it doesn't work out, they're responsible.. And further more, I actually believe that the future belongs to the young people.. And they often have much better ideas about what to do, then the more senior people do.. - Although, I think what I'm hearing from you though, is you were just this ultimate renegade.. - I've no idea.. - You should just call yourself, you're an unapologetic renegade.. And I think that probably answers your question, of she felt very good about the decisions that she was making.. Which is wonderful..

Yes.. - [Man In Audience] Primarily as a researcher, but how do you figure out what problems are worth solving, and worth spending half a decade or a decade on, because you seemed to have worked on very impactful problems, like we have not have been really happening that time frame, that they are going to be use for it, so how do you figure that out-- - So Barbara, you've been an incredible problem picker.. And figuring out where to point your career.. So how do you do that? - Oh, I have no idea.. (audience laughing) Look, I feel like I was lucky in a way.. So I got into the field early.. I mean of course, at that time, it was all very confusing, because nobody knew how to do anything.. But when I saw that program methodology problem, it was clear that was a hugely important problem.. When you can see a problem that's hugely important, that's a great thing to work on.. I happened to be on a panel with some Noble Prize winners, a couple of years ago..

And that question came up, how do you find problems to work on? And what they said is, what you do.. You pay a lot of attention to what's going on around you, and every time you read a paper, or listen to a talk, you ask questions.. So you say, what didn't they do? What's wrong? What could be better? And that way, you find things, somehow that helps you find things that are directions to go in.. I mean actually, that doesn't match my story very well, because, in fact, I saw Bob Khan pose a problem, but still, that's another thing I do, I'm a very critical reader.. And I also got very good at understanding what I didn't know.. Which is almost more important than knowing what you do know, because then you can see where the holes are in your reasoning.. And when I read papers, I see the holes in their reasoning.. And that often gives you an idea which direction you should go in.. But I don't know, I can't really answer that question.. I just look for good problems..

Don't do incremental work.. But look for things that look like they're gonna be important.. And then try to be honest in your research and really think deeply and figure out what you understand, and what has yet to be worked on.. (applause) (calm electronic music)..