WEBVTT

1
00:00:00.000 --> 00:00:01.260
Gurdip Singh: So yes, if

2
00:00:02.340 --> 00:00:03.149
Gurdip Singh: You if you want to

3
00:00:04.770 --> 00:00:05.490
Gurdip Singh: So,

4
00:00:06.930 --> 00:00:08.099
Gurdip Singh: Good morning everyone.

5
00:00:09.900 --> 00:00:26.490
Gurdip Singh: On behalf of NSF and sighs I welcome you to our Distinguished Lecture Series. Thank you all for joining. I know this is a busy time of the semester so appreciate you taking the time to to join.

6
00:00:27.930 --> 00:00:34.230
Gurdip Singh: It is with great pleasure to introduce our speaker today, Dr. Barbara cough.

7
00:00:36.030 --> 00:00:42.270
Gurdip Singh: Distinguished computer scientist and great inspiration to so many of us so

8
00:00:43.830 --> 00:01:01.830
Gurdip Singh: Barbara received her BSc in mathematics from University of California, Berkeley and a PhD in computer science from Stanford University in 1968 to one of the very first women among the first women to receive a PhD in computer science.

9
00:01:03.630 --> 00:01:21.000
Gurdip Singh: Subsequently, she worked at MITRE Corporation and then at MIT from 1972 onwards, where she is an institute professor and has held several leadership positions in the past.

10
00:01:24.360 --> 00:01:33.480
Gurdip Singh: Her he has research interests are in distributed in parallel systems programming methodologies programming languages.

11
00:01:34.980 --> 00:01:37.770
Gurdip Singh: She has a long list of accomplishments

12
00:01:39.540 --> 00:01:52.530

Gurdip Singh: Going to go to all of them. So among a few among those say she's a member of the National Academy of Engineering, the National Academy of Sciences. The National Inventors Hall of Fame.

13
00:01:52.980 --> 00:02:07.740
Gurdip Singh: And the messages. It's Academy of Sciences and very importantly for the science community. She received the ACM Turing Award in 2009 and the IEEE vendome and metal in 2004

14
00:02:09.330 --> 00:02:23.550
Gurdip Singh: So today, she's going to talk to us about our reflections on programming methodologies. And so before I turn it over to her. I just want to let our audience know that if you have questions.

15
00:02:24.960 --> 00:02:39.000
Gurdip Singh: Just feel free to type them into the chat session chat window as as the webinar progresses when once the presentation is completed, I will then go through those

16
00:02:40.170 --> 00:02:56.400
Gurdip Singh: Questions that have been posted in the chat window and that will be our mode of interaction. So, so thank you once again for joining. And again, it's a great pleasure that I asked. DR. BARBER he have to present her Distinguished Lecture.

17
00:02:58.770 --> 00:03:15.270
Barbara Liskov: All right, thank you for that introduction. Um, I was asked as part of my talk to talk a bit about my career. This truck actually has a lot built into it anyway. But I wanted to start if I can figure out how to get to the next slide, which I can't

18
00:03:16.350 --> 00:03:17.160
Barbara Liskov: Try this.

19
00:03:18.510 --> 00:03:33.660
Barbara Liskov: Oh, here we go. Sorry. Okay, so this repeats, a bit of what you heard, I grew up in San Francisco. I went to Berkeley in those days. I mean, Berkeley is a great school and in those days it was actually free to go to Berkeley.

20
00:03:34.980 --> 00:03:42.870
Barbara Liskov: I started off in physics, because I think I thought that was the hardest major but I realized pretty quickly that my

21
00:03:43.680 --> 00:04:00.330
Barbara Liskov: Physical intuition is not that great. So I switched to math. And I finished up with a BA. It was not a Bachelor of Science in math in 1961 and I applied, I should I should say, by the way, that there were almost no women in my classes I can remember.

22
00:04:01.440 --> 00:04:10.290
Barbara Liskov: One other woman in some of my math classes, but that was about it. This was not really the thing you

were supposed to be doing. If you were a woman at that period of time.

23
00:04:11.490 --> 00:04:15.720
Barbara Liskov: I applied for graduate school when I graduate. I got into a couple of places.

24
00:04:17.190 --> 00:04:20.070
Barbara Liskov: But I decided that I really wasn't ready

25
00:04:21.090 --> 00:04:26.190
Barbara Liskov: To make that kind of commitment and so I decided instead to get a job.

26
00:04:27.030 --> 00:04:34.830
Barbara Liskov: And so I moved to the Boston area that was where my father's family had come from. And I went there with a friend of mine who graduated from Stanford.

27
00:04:35.430 --> 00:04:47.130
Barbara Liskov: And when I arrived there. I looked for a job and I couldn't find a good job as a mathematician, which really isn't surprising because you need a lot more math than what I knew to do something interesting.

28
00:04:47.640 --> 00:05:00.390
Barbara Liskov: But I was offered a job as a programmer. I didn't even know computers existed at this point. And there were computers at Berkeley, but I was not in the engineering school and I really didn't know anything about them.

29
00:05:01.110 --> 00:05:07.050
Barbara Liskov: And they were hiring people like me in those days, because there was no computer science major.

30
00:05:07.440 --> 00:05:17.550
Barbara Liskov: And therefore, there wasn't a pool of train people that they could rely on. And so they would hire people like me who didn't know anything, but they thought maybe we could do the work.

31
00:05:18.060 --> 00:05:27.690
Barbara Liskov: And my first day on the job they handed me a Fortran manual and a little program to do. And they said, you know, go forth and write this program and so

32
00:05:28.680 --> 00:05:37.800
Barbara Liskov: I was entirely self taught in Fortran at that job I discovered something that I really, really liked. I was really, really good at it.

33
00:05:38.580 --> 00:05:48.840
Barbara Liskov: I worked at mitre for a year and then I switched to work at Harvard, and at Harvard, I was working on their machine translation project. So that was an early AI project.

34

00:05:49.230 --> 00:05:57.210
Barbara Liskov: Where they not easily believe that just in a few years, they were able to be translated from English to some other language, although at that point.

35
00:05:57.630 --> 00:06:01.080
Barbara Liskov: Their program couldn't even successful in our sentences.

36
00:06:01.830 --> 00:06:10.830
Barbara Liskov: None of this had anything to do with me because I was just a programmer and my job was to maintain a very large program written in assembler

37
00:06:11.190 --> 00:06:18.540
Barbara Liskov: So think about in those days it was all printouts think about a print out about two inches thick. That was the size of that program.

38
00:06:19.530 --> 00:06:31.110
Barbara Liskov: This was a great choice for me. I actually did it because I liked the commute better. But it turned out to be a great choice because now I got to learn how that machine work. It was a 7094

39
00:06:31.620 --> 00:06:41.610
Barbara Liskov: And so I then could understand what the Fortran Compiler had been doing and what was really going on with programs executed. Plus, I got to see

40
00:06:42.240 --> 00:06:47.340
Barbara Liskov: A very big program that I had to maintain and this taught me a lot about good programming practice.

41
00:06:48.060 --> 00:07:00.750
Barbara Liskov: I early in that time. It's at Harvard, I decided to apply to graduate school, because I was learning very fast, but it was clear that I could learn a lot faster. If I had somebody teaching me and so

42
00:07:01.470 --> 00:07:12.660
Barbara Liskov: I went to graduate school, I decided to go to Stanford, because I wanted to go back to the Bay Area in started there in 1963 and I went there without any support.

43
00:07:13.350 --> 00:07:24.210
Barbara Liskov: I didn't even really know there was such a thing, and my recollection is that walking up the steps on the very first day there I met john McCarthy and asked for support.

44
00:07:24.990 --> 00:07:32.970
Barbara Liskov: In retrospect, this seems highly unlikely that I would do that. But at any rate I did end up working with McCarthy, he did support me with an RA.

45
00:07:33.660 --> 00:07:41.280

Barbara Liskov: In retrospect, I suspect they thought I would be working in AI because of my work at Harvard, even though I wasn't a researcher in AI.

46
00:07:42.000 --> 00:07:57.120
Barbara Liskov: So Stanford was great. I think there were five people admitted the year I arrived. It wasn't even a computer science department, yet I realized partway through. And by the way, I was the only woman that your soo Graham came following a year.

47
00:07:58.410 --> 00:08:08.490
Barbara Liskov: I realized partway through that I really would prefer to be in computer systems, but I decided to stick with AI to get my PhD and then I would be able to go on and make changes in what I was doing.

48
00:08:09.390 --> 00:08:20.280
Barbara Liskov: When I graduated in 1968 and I didn't have any offers and good faculty position, which I naturally thought made me think, well, I'm probably not good enough.

49
00:08:20.880 --> 00:08:33.330
Barbara Liskov: Which is what you think when something like that happens to you. But at any rate, I decided to go back to mitre. Now, I wanted to go back to the Boston area because the man whom I married was living there and

50
00:08:34.920 --> 00:08:42.330
Barbara Liskov: But I went to my daughter and instead of being a programmer. I was now a researcher and this turned out to be a great thing because it mitre

51
00:08:43.380 --> 00:08:51.090
Barbara Liskov: mitre works for the government, as I'm sure you know, and they were doing research in system programming systems and they needed people to do this work.

52
00:08:51.360 --> 00:09:08.070
Barbara Liskov: And so my first day on the job. I was handed this great project to, first of all, implemented machine architecture in micro programming. And then I implemented a time sharing system to run on top of that. And then in about 1978 when that project was finished.

53
00:09:10.020 --> 00:09:23.340
Barbara Liskov: I was asked to look into the software crisis. So what is the software crisis, many people today don't know what it was. But the problem was, at that point in time we simply did not know how to build programs that worked.

54
00:09:25.050 --> 00:09:35.310
Barbara Liskov: And as a result, software development efforts failed, and it was very common in the 60s, the 70s, the 80s to pick up the newspaper and see a report.

55
00:09:35.640 --> 00:09:43.620
Barbara Liskov: About company x, who had spent millions of dollars and hundreds of man years. And in the end, they

just had to throw out the entire system.

56
00:09:44.160 --> 00:09:54.180
Barbara Liskov: Because it didn't work. Part of the problem was that they didn't have well trained people part of the problem was that they had a tendency to skimp on the hardware.

57
00:09:54.450 --> 00:10:03.600
Barbara Liskov: With the sort of naive idea that the software could make up the difference. There was very little understanding of what it meant to build software and how much work. It was

58
00:10:03.870 --> 00:10:13.500
Barbara Liskov: But at any rate. This was clearly a crisis, the government was very concerned because they were building quite advanced software systems. And so I was asked to think about what could be done about that.

59
00:10:14.580 --> 00:10:27.360
Barbara Liskov: I should say, by the way. The name of this talk is reflections. Because when I got the Turing Award in 2000. It was actually 2008 ago for some reason in 2010 although for some reason it's called 2009

60
00:10:28.410 --> 00:10:35.190
Barbara Liskov: I started reflecting on my career and I went back and thought about all these early years in programming methodology.

61
00:10:35.640 --> 00:10:42.090
Barbara Liskov: So here I am in programming methodology and so of course I did what any intelligent person does I start to read the literature.

62
00:10:42.510 --> 00:10:50.070
Barbara Liskov: And I discovered that program methodology was about two main topics, one was about design and the other was about program structure.

63
00:10:50.550 --> 00:10:55.230
Barbara Liskov: And I'm going to tell you about it, just a few of the papers I read some places are running

64
00:10:55.680 --> 00:11:02.220
Barbara Liskov: Programs for their undergraduates, where they have them read these old papers. There are many interesting old papers that students ought to know about.

65
00:11:03.030 --> 00:11:13.200
Barbara Liskov: So here's the first one. I'm sure you all know this and screw Dykstra, wrote, not a paper, but just a letter to the Communications of the ACM go true statement considered harmful.

66
00:11:14.100 --> 00:11:29.970
Barbara Liskov: And another thing you may not be aware of is that in those days programming was held in very, it was

not thought to be an interesting intellectual challenge and programmers were kind of dismissed as doing interesting intellectual work.

67
00:11:31.200 --> 00:11:37.050
Barbara Liskov: I actually saw this in fact even into the 80s when I would visit other departments where

68
00:11:37.740 --> 00:11:44.010
Barbara Liskov: The departments were controlled by mathematicians and they sort of dismissed the intellectual content of programming.

69
00:11:44.790 --> 00:11:50.610
Barbara Liskov: Director was partly making a point that programming is a difficult intellectual problem.

70
00:11:51.330 --> 00:11:58.710
Barbara Liskov: He's talking about the fact that in order to understand whether programs are working correctly. We need to reason about them.

71
00:11:59.100 --> 00:12:06.600
Barbara Liskov: And what we have in front of us is a piece of text, but what's actually happening is you have a program running. So you have to go from this

72
00:12:06.870 --> 00:12:17.220
Barbara Liskov: Thing in front of your face to thinking about the thing that's happening under the covers and the reason he thought to go to State and was bad was because it dis

73
00:12:17.700 --> 00:12:24.600
Barbara Liskov: disrupts the relationship between what you're looking on the text and what's happening in practice when the program is running.

74
00:12:25.110 --> 00:12:32.550
Barbara Liskov: And the way I like to think about this is that you're debugging a program, you get to a place where there's an error in order to understand

75
00:12:33.000 --> 00:12:39.330
Barbara Liskov: How you got there, you have to think about, you have to understand what's going on. You have to think about how you got there.

76
00:12:39.720 --> 00:12:48.090
Barbara Liskov: And if you have well structured programs with you know why how statements if statements and procedure calls it's relatively easy to do this.

77
00:12:48.390 --> 00:12:59.160
Barbara Liskov: But if you have a program with go to send it. And these were sometimes referred to as bowls of

spaghetti, because it was like you had pointers everywhere. It can be very, very hard to figure it out.

78
00:13:00.270 --> 00:13:11.820
Barbara Liskov: This paper was very controversial of partly because people thought that without go tues maybe they couldn't write the programs they needed to write

79
00:13:12.120 --> 00:13:25.800
Barbara Liskov: Because programming languages were pretty primitive in those times compared to what we have today and their control structures were somewhat lacking, but the other one was that Dykstra tended to annoy people and people were very annoyed because they said

80
00:13:26.820 --> 00:13:36.930
Barbara Liskov: I write very good programs and I use go tues and that is absolutely correct. You can write excellent programs in any lousy programming language and you can write terrible programs.

81
00:13:37.140 --> 00:13:49.020
Barbara Liskov: And even a very good programming language, but nevertheless Dykstra was absolutely right to go to save it was not a good idea. And we were able to manage just fine without it. Okay, use me

82
00:13:51.360 --> 00:13:58.410
Barbara Liskov: The second papers by Nicholas fear. This is about design top down design, which was also a topic of interest at the time.

83
00:13:59.610 --> 00:14:13.920
Barbara Liskov: And Nicholas had a an idea that you're sort of wrote an abstract program. This is like what we think of today as computational thinking you know you bribed the program into tasks and you pick up one of those tasks and think about it some more.

84
00:14:15.210 --> 00:14:27.540
Barbara Liskov: And his example was the Queen's problem. He said, let's think about this problem as adding the next queen to the next column. So you have a solution that works for him columns you tried to add the queen to the next one.

85
00:14:27.810 --> 00:14:33.540
Barbara Liskov: If you can do it, you go forward. Otherwise you backtrack and and continue that way. That was his example.

86
00:14:35.160 --> 00:14:48.720
Barbara Liskov: We know today that top down design really is the way to go because after all, if you don't think about the problem you're trying to solve. I don't see how you can possibly actually solve it, but of course it's not a simple thing where you can just sort of March, right down like that.

87
00:14:50.370 --> 00:15:02.970
Barbara Liskov: Of the third paper. And the last one is one by Dave parness parness wrote a couple of papers about this and and now he's getting into this issue of modularity, which I think is really the key issue in programming

methodology.

88
00:15:04.020 --> 00:15:13.380
Barbara Liskov: And he what he said in another paper was there are these things called modules, we know we want our programs to build out of them. But we don't know what they are.

89
00:15:14.730 --> 00:15:22.020
Barbara Liskov: And that was the problem at the time in this paper. He talked about the connections between modules and the point he's making here.

90
00:15:22.410 --> 00:15:33.990
Barbara Liskov: Is that when people thought about these modules, whatever they were they neglected all sorts of connections. So, for example, they might think about a procedure is a module. In fact that was basically the only module mechanism, they had

91
00:15:34.410 --> 00:15:43.860
Barbara Liskov: But they might neglect the fact that the procedure was communicating with the outside world via a whole bunch of global variables. And so he saying you have to capture everything

92
00:15:44.280 --> 00:16:03.540
Barbara Liskov: About what the module is doing in order to be able to make modularity work. Okay, so I read these papers I read a whole bunch of others. And I realized that I had invented. Oh, excuse me, I want to stop for a minute and just talk about modularity today so

93
00:16:04.770 --> 00:16:16.230
Barbara Liskov: Here's what we know today a program is a collection of modules, each module has an interface described by specifications. So here we're talking about day Parnassus thing.

94
00:16:17.130 --> 00:16:25.260
Barbara Liskov: And we understand that that interface had better be a complete the complete way you interact with a module in the specifications and better describe all the behavior.

95
00:16:25.650 --> 00:16:35.790
Barbara Liskov: And an example of a module sort routine where the interface is the arguments and results and the specification says when you return the array is in sorted order.

96
00:16:36.840 --> 00:16:47.910
Barbara Liskov: So it's a difference between what the module does and how it doesn't. There's nothing in that description about the fact that I'm using whatever sort routine. I chose to use

97
00:16:51.210 --> 00:16:52.200
Sorry, I'm having

98

00:16:53.550 --> 00:16:54.030
Okay.

99
00:16:56.280 --> 00:17:04.440
Barbara Liskov: And then we know now that we can talk about correctness images implementation correct if it meets specifications and

100
00:17:07.050 --> 00:17:22.050
Barbara Liskov: This gives us local reasoning, which is the powerful thing we're looking for, provided we can be certain that the rest of the code depends only on the specification and doesn't. In fact, somehow or other interact with the module in some other way.

101
00:17:25.830 --> 00:17:29.460
Barbara Liskov: So actually, in 1970 when I was working at

102
00:17:31.380 --> 00:17:38.400
Barbara Liskov: On this, we knew that we wanted modules. We didn't understand what they were we understood the benefits the local reasoning.

103
00:17:38.820 --> 00:17:45.930
Barbara Liskov: And independent development was maybe the most important thing at that time because you had teams of programmers and you had to give them different things to work on.

104
00:17:47.190 --> 00:17:52.170
Barbara Liskov: It, we may not have understood the Modify ability quite so well. But what that means is we understood that.

105
00:17:52.920 --> 00:17:59.580
Barbara Liskov: You might have to change a piece of the program. And if he hadn't developed correctly, then you could rip out that piece, replace it with another

106
00:17:59.880 --> 00:18:05.940
Barbara Liskov: And the system as a whole would continue to work the problem was we had absolutely no idea with the modules were

107
00:18:06.390 --> 00:18:15.750
Barbara Liskov: procedures were all that we understood and these are nowhere near powerful enough to build systems and people didn't get the connections right as harness was saying.

108
00:18:16.380 --> 00:18:27.750
Barbara Liskov: Okay, so I thought about all these papers and a bunch of others. And I realized that I had invented assignments for the I'm not a design methodology. That's what I call it, but more like a modularity idea.

109

00:18:28.410 --> 00:18:39.990
Barbara Liskov: And this paper was actually about design. Also, but I'm going to talk about the modularity and this was when I was working on that time sharing system that I had developed in the previous project.

110
00:18:41.010 --> 00:18:44.940
Barbara Liskov: Time sharing was a new idea at the time and

111
00:18:46.110 --> 00:18:59.370
Barbara Liskov: I was very concerned, I had a small group of programmers about how we were going to be able to produce this fairly complicated system in a way that was effective. And the way programs were built at the time.

112
00:19:00.450 --> 00:19:09.840
Barbara Liskov: There was an awful lot of communication through global variables. So imagine that there's some huge global stage and then you had these modules.

113
00:19:10.230 --> 00:19:21.030
Barbara Liskov: And whatever they are, and they're all interacting with one another to that global state. And this is really not a very good idea. So what I decided to do in the development of this venous system.

114
00:19:21.930 --> 00:19:31.050
Barbara Liskov: Was to break up the global state into what I called partitions each partition owned a part of the global stage and the rest of the program.

115
00:19:31.620 --> 00:19:41.520
Barbara Liskov: Did not access that stage so that state was local to the partition and in order to provide access the partition provided operations procedures.

116
00:19:41.940 --> 00:19:48.240
Barbara Liskov: That could be called by the rest of the program and only those procedures access to that stage.

117
00:19:49.200 --> 00:20:01.200
Barbara Liskov: So what you see here actually is a very modern way of looking at modularity and it was new at the time. I wasn't inventing a methodology. At the time, I was just trying to get that system to run

118
00:20:03.180 --> 00:20:03.630
OK.

119
00:20:05.130 --> 00:20:15.000
Barbara Liskov: OK, so now it's 1972 and I moved to MIT. So what happened was I wrote a paper on Venus, the

120
00:20:16.080 --> 00:20:25.440
Barbara Liskov: Time sharing system that I had developed in my previous project. At MITRE, and by the way, I should stop for a minute and say, the fact that I went to mitre was really

121
00:20:26.250 --> 00:20:37.650
Barbara Liskov: A gift because I was able to make that move from AI to systems without all the added pressure, I would have had in a university job of teaching courses and so forth.

122
00:20:39.480 --> 00:20:47.940
Barbara Liskov: So, you know, it's like a door closed and another door opens and that actually has been my experience in my career mean that's what happened to me.

123
00:20:48.330 --> 00:20:53.190
Barbara Liskov: When I didn't get a job in math and therefore I got into computer science instead

124
00:20:53.670 --> 00:20:59.940
Barbara Liskov: And you know if I'd gone to university, I don't know what would have happened. But this way I felt like I had a candy box at

125
00:21:00.330 --> 00:21:07.860
Barbara Liskov: mitre I had these wonderful projects and I was just doing them and learning a ton of stuff. Anyway, I had written a paper on Venus.

126
00:21:08.610 --> 00:21:19.350
Barbara Liskov: In 1971 submitted it to SSP and which is, as you know, the top systems conference and it was a prize paper. So sp and meanwhile

127
00:21:19.980 --> 00:21:32.520
Barbara Liskov: Title nine was about to pass and Title nine, as you know, had to do mostly with athletics access for women to athletics, but it was having an impact on access for women.

128
00:21:33.030 --> 00:21:43.230
Barbara Liskov: To things in universities in general. And so some universities and MIT was one of them. I would say a handful in 1972

129
00:21:44.070 --> 00:21:52.560
Barbara Liskov: Decided they were looking for women on the faculty and sitting in the audience at my talk at SSP was Corby Fernando Corbett show

130
00:21:53.070 --> 00:22:02.550
Barbara Liskov: And Jerry Salter was professor at MIT. He was the head of my session and they were they were looking because Jerry Wiesner who was the president of MIT.

131
00:22:03.030 --> 00:22:16.920
Barbara Liskov: had told them he was interested, this kind of stuff does come from the top. And so I was invited to apply to MIT. And I went there in the fall of 1972 and here's a kind of a funny story.

132
00:22:18.150 --> 00:22:21.420
Barbara Liskov: God, I can't remember his name. The internet guy.

133
00:22:23.160 --> 00:22:30.480
Barbara Liskov: I'll think of his name. Anyway, he tells a joke about how I got his job. He applied to MIT. They gave me the job instead

134
00:22:30.810 --> 00:22:40.980
Barbara Liskov: I mean, look how badly. He ended up in the end. And sometimes I'll think of his name later. But this is what happened. So should get old. Okay, so I moved to MIT in 1972

135
00:22:41.430 --> 00:22:53.160
Barbara Liskov: And honestly, this was a perfect time for me to move because at monitor the way it worked was we looked for calls for proposals from the government and

136
00:22:53.610 --> 00:22:59.940
Barbara Liskov: So, you know, I would have worked on program methodology for a bit. It's not fair. I would have continued working on it because something else would have come up

137
00:23:00.480 --> 00:23:13.830
Barbara Liskov: But at this point, I was really hooked programming methodology was a very important topic and I was just really interested in it, and my what I was thinking about was this question about

138
00:23:15.060 --> 00:23:24.180
Barbara Liskov: We had all these really good papers and they would describe a way of doing something and they would give you an example. And you read that paper and you think

139
00:23:24.600 --> 00:23:34.530
Barbara Liskov: Boy, that really is the right way to do it. But then when you came to the programming were working on and you tried to apply those ideas you just kind of fell apart.

140
00:23:35.220 --> 00:23:44.790
Barbara Liskov: And I think maybe it fell apart, partly because it just wasn't very well defined. It was a lot of hand wavy. So I was thinking about

141
00:23:46.740 --> 00:23:53.220
Barbara Liskov: You know, what can we do about this, and my way of working, by the way, is I work nine to five.

142
00:23:53.760 --> 00:23:59.400
Barbara Liskov: More or less. Maybe it's, you know, five or six or whatever, very intensely, and then I stopped for the day.

143
00:23:59.820 --> 00:24:06.720
Barbara Liskov: But that doesn't mean and I developed this in my year at mitre after my undergraduate program when

144
00:24:07.080 --> 00:24:19.050
Barbara Liskov: I decided I discovered how effective you can be, if you really use those hours and how useful it is to not work because this gives you a chance to rest. And then you come back to your work, rest of the next morning, plus

145
00:24:19.800 --> 00:24:23.520
Barbara Liskov: Your subconscious is still working and I'm still thinking about stuff and so

146
00:24:24.360 --> 00:24:34.080
Barbara Liskov: Every day I might go home with some problem I've been considering and overnight, maybe it got solved in the morning when I was thinking about

147
00:24:34.560 --> 00:24:51.390
Barbara Liskov: What am I going to do today. I often discovered I had a solution to a problem. I hadn't figured out how to solve yesterday. So anyway, at some point. Somehow I got the idea of abstract data types and it seems kind of obvious in I did it again. Okay.

148
00:24:52.800 --> 00:25:01.500
Barbara Liskov: Seems kind of obvious in retrospect I sort of saw that you could think of partitions as data abstractions. So here's the partition.

149
00:25:02.040 --> 00:25:09.750
Barbara Liskov: It looks kind of like an object, but really it's not thinking about it as the the program, the class that implements the data type.

150
00:25:10.170 --> 00:25:15.270
Barbara Liskov: That provides you with a bunch of operations, the representation of the objects is hidden

151
00:25:15.690 --> 00:25:21.390
Barbara Liskov: And some of these operations are constructors and others are the operations that you use to interact with the objects.

152
00:25:21.690 --> 00:25:38.550
Barbara Liskov: But nobody had made this connection yet. So I had this wonderful you know moment when all of a sudden I saw that I could make this connection and I understood immediately that this was very important. And the reason it was very important is because it links modularity to design

153
00:25:40.620 --> 00:25:52.410
Barbara Liskov: Because the way we designed a spike in inventing abstractions. In fact, when I was teaching my course and how to write big programs, I would talk about a methodology that said

154
00:25:52.860 --> 00:26:01.860
Barbara Liskov: invent an abstract machine with all the operations. I eat the procedures and all the data types I eat the data abstractions that you would like to have

155
00:26:02.250 --> 00:26:14.130
Barbara Liskov: And then you can just implement your program using that abstract machine and then you pick up one of those abstractions and continue going, but they're abstractions. They're not just pieces of code. Their abstractions and

156
00:26:14.820 --> 00:26:20.100
Barbara Liskov: And that was really important. And furthermore, I didn't think programmers would have any trouble.

157
00:26:20.430 --> 00:26:27.630
Barbara Liskov: With data types because they already understood how to invent procedures so they understood procedural abstractions. This is just another one.

158
00:26:28.020 --> 00:26:37.500
Barbara Liskov: And starting to think about what kind of abstract data do I need that wouldn't be an issue for them. They understood data types already in fact of course there were data types.

159
00:26:38.220 --> 00:26:44.130
Barbara Liskov: data abstractions in their higher level programming languages that were implemented by the compiler and

160
00:26:45.000 --> 00:26:51.360
Barbara Liskov: But in order for this to work. We're going to need something in programming languages. So I started to think about programming languages.

161
00:26:52.290 --> 00:27:02.520
Barbara Liskov: And I started to work on this with a with Steve zealous who at that point was a graduate student at MIT. He was also an IBM employee and he had had

162
00:27:03.210 --> 00:27:10.770
Barbara Liskov: A similar idea. And so we decided to see what we could do with it. And what we're doing now is we're trying to think about what would it mean

163
00:27:11.490 --> 00:27:20.340
Barbara Liskov: To have a programming language that supports of abstract data types. And of course, we read the literature, which is pretty well positioned because I

164
00:27:21.030 --> 00:27:25.980
Barbara Liskov: Was very familiar with list since I had done my whole thesis and list.

165

00:27:26.520 --> 00:27:37.050
Barbara Liskov: And Steve came from IBM and so he was up on all those IBM languages, which were the major languages at the time. And of course, we knew about other languages like the alcohol family and so forth.

166
00:27:37.710 --> 00:27:43.590
Barbara Liskov: Okay. So Stephen I read the literature and I'm just going to mention a couple things. This is an early paper on simulate

167
00:27:45.660 --> 00:27:54.510
Barbara Liskov: Clearly, a very important piece of work we looked at this, we felt that its main point had to do with hierarchy and

168
00:27:54.930 --> 00:28:08.820
Barbara Liskov: We thought that was kind of not only were interested in. Plus, it had no encapsulation. So this idea that you limit access to internal state to just the module that wasn't in there. And also, we didn't go in that direction.

169
00:28:10.440 --> 00:28:16.590
Barbara Liskov: I'm not going to tell you very about very many papers at that point. This one we found very interesting Jim Morris.

170
00:28:17.670 --> 00:28:27.990
Barbara Liskov: Protection in programming languages and what Jim was doing there was, he was defining rules for modularity and what he said in that paper is that

171
00:28:29.310 --> 00:28:41.850
Barbara Liskov: In order for modularity to work the first rule is that code outside of module must not manage the mass not access in it or must not modify the data managed by the module.

172
00:28:42.600 --> 00:28:49.020
Barbara Liskov: And that's clearly necessary because if you want, what you want for modularity independent reasoning.

173
00:28:49.980 --> 00:28:56.160
Barbara Liskov: You've got to have a barrier around the modules, this is going to work if some code on the outside, you can get in there and muck with your internal state.

174
00:28:56.520 --> 00:29:05.880
Barbara Liskov: Then your reasoning isn't going to work at all. So that was the first point, Jim was making. He also pointed out that really the code on the outside shouldn't even look at your internal state.

175
00:29:06.330 --> 00:29:17.550
Barbara Liskov: And this is important. It's really important for it is two ways of thinking about it if the if the internal state is observed this major specification ought to be covering it.

176

00:29:18.300 --> 00:29:26.430
Barbara Liskov: But another way of looking at it is your specifications. Okay, just better protected because you want to be able to replace this module.

177
00:29:26.970 --> 00:29:33.090
Barbara Liskov: This implementation with another one if for some reason discover it's not good. Like, it isn't implemented fast enough.

178
00:29:33.540 --> 00:29:39.300
Barbara Liskov: And if code on the outside could have observed your internals. You are much more limited in what you can do.

179
00:29:39.960 --> 00:29:46.890
Barbara Liskov: Okay, so we read them. However, Jim, then went on to say, well, how are we going to manage this and he suggested something akin to encryption.

180
00:29:47.820 --> 00:29:55.380
Barbara Liskov: You certify encrypt all objects coming out of the module and when they come back in. You can tell that they've been mucked with

181
00:29:55.800 --> 00:30:07.470
Barbara Liskov: And so therefore, you're in control. But of course, this is obviously not a solution that's going to fly in practice. Okay. So Steve, and I thought about this and we worked on this for maybe only six or eight months.

182
00:30:08.730 --> 00:30:20.130
Barbara Liskov: And our motive record of working was we met every day at lunchtime, or almost every day and my recollection of the summer of 1973 was that

183
00:30:21.000 --> 00:30:26.610
Barbara Liskov: It was sunny every day in Cambridge, Massachusetts, which obviously wasn't true, but we met outside the lovely area.

184
00:30:27.090 --> 00:30:37.620
Barbara Liskov: And then we would think independently, and by the end of that summer we had written up our ideas and submitted it to a conference that I don't believe exists anymore, but was about programming language research at the time.

185
00:30:38.460 --> 00:30:43.920
Barbara Liskov: And this program made a big splash because this was an idea whose time had come.

186
00:30:45.120 --> 00:30:58.080
Barbara Liskov: And basically what it did was it was a sketch of what the programming language needed to do and we were focused on the idea that we need a language and a compiler that's going to enforce the necessary rules at compile time.

187
00:30:59.130 --> 00:31:11.010
Barbara Liskov: Okay, so now it's the fall of 1973 and I took the obvious next step of, oh, you know, I forgot to say something about MIT. I'm going to go back for a minute.

188
00:31:11.910 --> 00:31:28.770
Barbara Liskov: I meant to tell you that when I arrived at MIT. So I did tell you that title nine came along and that its impact was that universities were looking for women. And in fact, when I had applied for faculty positions in 1968

189
00:31:30.450 --> 00:31:38.760
Barbara Liskov: there weren't very many women in faculty positions at universities and they really were not looking for women and maybe this was particularly true in STEM.

190
00:31:39.210 --> 00:31:51.240
Barbara Liskov: But I think it was true across the board. And when I got to MIT in 1972 there were 10 women on the faculty of the faculty of almost 1000 people

191
00:31:52.200 --> 00:32:03.660
Barbara Liskov: I was the first woman in computer science, computer science wasn't a department, we didn't even it was in the double E department and later the W department became

192
00:32:04.380 --> 00:32:16.380
Barbara Liskov: Electrical engineering and computer science which it is still today. But at that point it was just double E. And there was one other woman in my department Millie dress will house, a very distinguished physicist.

193
00:32:17.520 --> 00:32:26.880
Barbara Liskov: I was the first one in computer science. And one of the things I noticed in a couple of years after I joined the MIT was quite a number of women.

194
00:32:27.150 --> 00:32:41.010
Barbara Liskov: were added to the faculty who had previously been in research position. So, I mean, this is what you know really confident women had been doing before things started to change. OK, so now continuing on my way here.

195
00:32:42.210 --> 00:32:47.820
Barbara Liskov: I was working on clue, and I did this work primarily with three graduate students who

196
00:32:49.260 --> 00:33:01.080
Barbara Liskov: Showed up in my office is the way I think about them in the fall of 1973 wanting to work with me. It wasn't quite that way. I mean, Russ Atkinson, I talked to him later.

197
00:33:01.890 --> 00:33:16.830
Barbara Liskov: And he said, Well, what really happened was I was my you were my academic advisor that man I advise them about what courses to take and you suggested that I joined your research group. So, you know. Anyway,

these three students were the main ones.

198
00:33:18.570 --> 00:33:24.270
Barbara Liskov: It was quite a large research group though me a lot of people came to our design meetings we had weekly design meetings.

199
00:33:24.720 --> 00:33:34.530
Barbara Liskov: In this picture, that's me and around 1974 and that's jack Dennis, who was a professor at MIT. And he was one of the people that helped me

200
00:33:35.490 --> 00:33:46.620
Barbara Liskov: In fact, I wrote my first NSF proposal at that time. And I think jack was a co author and he certainly helped me write the proposal and jack used to come to our weekly meetings.

201
00:33:48.360 --> 00:33:58.380
Barbara Liskov: And of course, Steve was still involved. Although Steve was now working on his PhD thesis, which was about algebraic specifications for data types. And then there were another a number of other people who were

202
00:33:59.310 --> 00:34:05.280
Barbara Liskov: As time went by more people join my research groups and but others were interested and they would come to the weekly meetings.

203
00:34:06.390 --> 00:34:07.380
Barbara Liskov: Okay, so

204
00:34:09.660 --> 00:34:23.280
Barbara Liskov: But the three graduate students. I mentioned they were the main designers and it was clear that the next step, had to be to design a programming language. But here's my rationale and I had a well thought out rationale

205
00:34:24.480 --> 00:34:30.150
Barbara Liskov: The nice thing about a programming language is Scott rules, it has to be precise. It is a mathematical object.

206
00:34:30.450 --> 00:34:37.290
Barbara Liskov: Even though in those days it was described by a programming language manual, which often happens stakes in it.

207
00:34:37.530 --> 00:34:45.330
Barbara Liskov: Or ambiguities and so the first compiler would resolve the ambiguities in one way, and the second would do it differently. And so there was a mess.

208
00:34:45.660 --> 00:34:52.890
Barbara Liskov: Because this was before we had formal specifications and so forth and so on. But nevertheless,

compared to a paper in the literature.

209
00:34:53.310 --> 00:35:00.960
Barbara Liskov: That showed you an example and wages hands. This is much more precise. And the other thing is programming language is a tool.

210
00:35:01.590 --> 00:35:05.460
Barbara Liskov: And you have to understand what the tool, whether it really works. So

211
00:35:06.030 --> 00:35:15.570
Barbara Liskov: You know, it's important that it be convenient to use. It's important signal sufficiently simple, it does the things that needs to do needs to have the right expressive power.

212
00:35:16.080 --> 00:35:24.750
Barbara Liskov: And has to have good performance. It doesn't have to have huge performance. Performance. This is overrated. But it has to be good enough that you don't mind using it so

213
00:35:25.020 --> 00:35:36.510
Barbara Liskov: By designing and implementing a programming language. We're figuring out what programming what abstraction really was data abstraction. And we're also making sure that it would work in practice, and although it seems obvious today.

214
00:35:36.990 --> 00:35:39.600
Barbara Liskov: It was not always the time that this was going to really work.

215
00:35:41.280 --> 00:35:48.900
Barbara Liskov: And I'm not going to tell you much about clue. But I just want to point out some facts about it. So you have a sense of the world at that time.

216
00:35:50.280 --> 00:35:53.850
Barbara Liskov: And first of all, we were absolutely determined to do compile time type checking

217
00:35:55.770 --> 00:36:02.220
Barbara Liskov: And. And actually if you look at this slide, mostly what you see as a kind of a love hate relationship with list so

218
00:36:03.210 --> 00:36:12.180
Barbara Liskov: I did my thesis. It was chess games in list and I was always so furious when I would have to discover an error in the program by debugging.

219
00:36:12.630 --> 00:36:22.680
Barbara Liskov: That could have been caught by a compiler. If only we had compile time type checking. So I thought, really, it would be great to have something that enforces the rules and, furthermore,

220
00:36:23.280 --> 00:36:31.290
Barbara Liskov: With this whole new idea of data abstraction, we need to get our head around what it meant to make sure that it worked. Okay so static type checking heat based

221
00:36:31.680 --> 00:36:40.200
Barbara Liskov: That's list and this may surprise you, looking back today, but in those days, there was a lot of controversy about Pointers. Pointers are evil.

222
00:36:40.650 --> 00:36:50.490
Barbara Liskov: You shouldn't have them. Don't want to heat ridiculous and Bill wolf and Mary Shaw. We're working on our farm at the same time, this was the other

223
00:36:51.300 --> 00:36:59.010
Barbara Liskov: Programming Language research project that was investigating data abstraction and they were using a non heat based approach.

224
00:36:59.340 --> 00:37:03.510
Barbara Liskov: And actually it caused him a big problem because the thing about data abstraction.

225
00:37:03.840 --> 00:37:11.820
Barbara Liskov: Is it has each the data type has objects, you don't actually want to think about how they're implemented at the time that you're defining what the time means

226
00:37:12.240 --> 00:37:23.670
Barbara Liskov: And that means you don't actually know how big they are, and with a heat based approach. No problem. You know, all you have in the stack is a pointer and you don't have to worry about how much space. It takes on the heat.

227
00:37:24.090 --> 00:37:30.630
Barbara Liskov: But if it's in the stack. This is a big pain in the neck and it cost them a lot of trouble separate compilation. This is listen to

228
00:37:31.200 --> 00:37:37.620
Barbara Liskov: Because that was just the way you did it. You wrote one procedure at a time. You just kept going. And that was also very free

229
00:37:38.070 --> 00:37:52.470
Barbara Liskov: And then a bunch of stuff I didn't do, because when you're in gauge internet in a challenging research project you throw out as much as you can. I decided no concurrency jack Dennis would have liked to have seen concurrency, but I decided that

230
00:37:53.550 --> 00:38:05.190

Barbara Liskov: We had enough on our plate without worrying about that too. No. Go tues because I thought my sister was right no inheritance, because that seemed like a distraction. Okay. Now, just one more slide about clue.

231
00:38:07.200 --> 00:38:11.340
Barbara Liskov: It took us several years to design it and implement it because

232
00:38:12.360 --> 00:38:26.250
Barbara Liskov: We had to innovate in many more ways than you might have expected this again has to do with what programming languages were like if the time. First we invented a mechanism for implementing data types. Those are the clusters. And that's where the name comes from

233
00:38:28.410 --> 00:38:39.720
Barbara Liskov: But then we had to face up to polymorphous so at that point in time programming languages didn't have any notion that you could write a piece of code, and it would work for many different types

234
00:38:40.320 --> 00:38:50.520
Barbara Liskov: And it kind of worked okay when you didn't have data abstraction. Because what your programming language had in it was just a very small set of types. And so maybe this wasn't a big deal.

235
00:38:50.910 --> 00:39:01.410
Barbara Liskov: Although, honestly, it must have been annoyed if you had to rewrite your sort routine because it had to do now with arrays of rules, instead of a race of integers.

236
00:39:01.860 --> 00:39:09.840
Barbara Liskov: But anyway, as soon as you have data abstraction. And you can see new types marching down the road, you know, you're going to have to write code.

237
00:39:10.350 --> 00:39:16.290
Barbara Liskov: That handles many types. And not only that, but you want your data abstractions themselves to be polymorphous you want to have a set

238
00:39:16.710 --> 00:39:25.290
Barbara Liskov: You know, not a set of integers, not a set of rules. You want to be able to define the set once and then have it worked for many different types, but it wasn't an officer, how to do this.

239
00:39:25.590 --> 00:39:37.230
Barbara Liskov: And it took us quite a while to figure it out. And we came up with a mechanism that was mostly but way ahead of its time, like what they have in Haskell now the the Haskell type classes.

240
00:39:38.340 --> 00:39:48.000
Barbara Liskov: And then we needed to deal with iteration, because many data types of collections. And when you have collections. You want to iterate over them without destroying them. And so we invented integrators

241

00:39:48.240 --> 00:39:59.130
Barbara Liskov: Most like what are called generators or rather, they are called generators. I forget the modern terminology that was a very nice mechanism. It was not a data abstraction. It was more like a procedure.

242
00:40:00.060 --> 00:40:07.350
Barbara Liskov: And we had to deal with exception handling at that point, people didn't know what exception handling should be like there were various proposals.

243
00:40:07.800 --> 00:40:14.250
Barbara Liskov: John good enough wrote a paper in 1974 I think was talking about all the different things that were available.

244
00:40:15.210 --> 00:40:24.900
Barbara Liskov: You know, there was a question about, you have the resumption model versus the termination model and we had to sort of way through all of this and we came out with a very nice mechanism.

245
00:40:25.500 --> 00:40:37.950
Barbara Liskov: There were other issues that we had to worry about lockdown. You know, what is the normal case. And how do you write your code and so forth. Anyway, they deal with all this stuff. We had to innovate in many different ways. All right, so we finished clue.

246
00:40:39.000 --> 00:40:43.620
Barbara Liskov: In the late 70s. I showed you the paper didn't show me the paper. I think I can.

247
00:40:45.600 --> 00:40:49.770
Barbara Liskov: If I didn't, didn't which came out I think around 1977

248
00:40:50.580 --> 00:41:05.130
Barbara Liskov: By then we had the language defined, we might have still been working on various implementations. We use the usual trick of bootstrapping through a Lisp variants. And then after that we both the compiler and flu and we had users. And so we had a

249
00:41:06.690 --> 00:41:21.870
Barbara Liskov: You know, we were up there and going concern and it was time to to figure out a new research project and I seriously thought about, well first I thought about programming language research but I decided that I didn't have any great ideas. So I didn't want to go on in that direction.

250
00:41:22.920 --> 00:41:36.450
Barbara Liskov: I even thought about commercialization, but it was different in those days, I would have had to, you know, form a company, it would have had to spend all this time maintaining code that was being used elsewhere. It wasn't a research path.

251
00:41:37.680 --> 00:41:47.010
Barbara Liskov: And so I was looking around and I read a paper by Bob Kahn about the ARPANET, which was still a

fairly new thing in those days.

252
00:41:47.490 --> 00:41:58.380
Barbara Liskov: And he talked about the dream of distributed programs or distributed computing where you could have a program that had components running at many different machines.

253
00:41:58.950 --> 00:42:09.540
Barbara Liskov: And wouldn't that be great but nobody really knew how to do it. So I thought, oh, well, there's like great research project. And so I jumped into distributed computing

254
00:42:10.020 --> 00:42:18.720
Barbara Liskov: And what I have on the slide or some of the things I accomplished, but I'm not going to talk about those today. It's a fascinating field distributed computing

255
00:42:19.980 --> 00:42:22.470
Barbara Liskov: But I continue to work on Turkey methodology.

256
00:42:24.390 --> 00:42:30.750
Barbara Liskov: Oops, let me go back just hang on just a minute. I'm going to go back a couple slides.

257
00:42:32.790 --> 00:42:49.680
Barbara Liskov: And but I didn't think about what I was doing as research. I was doing it mainly in the context of a course that I was development in it since it's MIT doesn't have a name. It has a number six 170 and it was our second course in computing

258
00:42:50.880 --> 00:43:01.230
Barbara Liskov: So the idea was the kids already knew how to write small programs. And now we were going to tell them how to write big programs. So it was really appropriate methodology course.

259
00:43:01.590 --> 00:43:12.000
Barbara Liskov: And I taught them about module program design I taught them how to reason about correctness. I taught them about specifications much of this was done jointly with john de tag.

260
00:43:12.630 --> 00:43:27.750
Barbara Liskov: John had done his PhD thesis on abstract model specs know maybe he did I forget specifications for abstract data types and, you know, we had the students writing specifications. We had them implementing

261
00:43:30.060 --> 00:43:35.490
Barbara Liskov: The abstraction function and the rep invariant. I mean, we're teaching them all this stuff and so

262
00:43:36.720 --> 00:43:42.390
Barbara Liskov: I, as I said, I continue to think about this. We talked, we talked them a top down design methodology and so forth.

263
00:43:43.500 --> 00:43:55.920
Barbara Liskov: And so the only thing I'm going to tell you about is one thing that came along in the program methodology area and that was the stuff about type hierarchy. So in 1987 I believe

264
00:43:56.940 --> 00:44:02.730
Barbara Liskov: I was asked to give the keynote at oops LA. So, whoops. There was a new conference there.

265
00:44:03.360 --> 00:44:15.240
Barbara Liskov: And another thing about what was going on and research at that time, which seems kind of odd today, but was true, then, is the East Coast and the West Coast were very far apart and

266
00:44:16.200 --> 00:44:28.080
Barbara Liskov: What was going on out there was different from what was going out in the east. So in the East Coast. We are working on data abstraction on the West Coast. They working on small talk and small talk.

267
00:44:28.500 --> 00:44:36.570
Barbara Liskov: Ronnie's so they were over in the inheritance area. And when I was invited to give this keynote at oops la

268
00:44:37.140 --> 00:44:43.260
Barbara Liskov: I thought, oh, well, this is a good opportunity for me to look into all that work that had been going on the west coast.

269
00:44:44.070 --> 00:44:56.010
Barbara Liskov: And so I started reading lots of papers on small talk and other object oriented languages that were being designed based on the work that had been done on small talk. And of course, small talk is based on Simul 67

270
00:44:56.880 --> 00:45:12.570
Barbara Liskov: And so what I discovered when I read these papers was that, as you know, small talk provided inheritance and of course it was the predecessor of the modern languages we use today, just like my work on clue was a predecessor and

271
00:45:13.860 --> 00:45:24.360
Barbara Liskov: And people were interested in inheritance, but it was used for two different purposes, it was us first of all to implementation techniques. So if I have an implementation.

272
00:45:24.750 --> 00:45:36.480
Barbara Liskov: Of something some data type, you know, like Windows, then I could have a subtype me a subclass that implements red windows by just sort of borrowing from the code and adding a little extra stuff.

273
00:45:37.320 --> 00:45:48.000
Barbara Liskov: And I have never actually been all that interested in that stuff though I don't deny that seems to be

useful for various things. One of the things that struck me, though in reading the papers on

274
00:45:49.110 --> 00:45:54.840
Barbara Liskov: These languages. These auditory languages, the way people didn't talk about specifications at all.

275
00:45:55.260 --> 00:46:00.120
Barbara Liskov: They were talking about the meaning of a subclass by explaining its differences in code.

276
00:46:00.510 --> 00:46:08.550
Barbara Liskov: What was going on and superclass. So that was kind of interesting and very different from what we were doing in six 170 where we were working on specifications.

277
00:46:09.240 --> 00:46:15.030
Barbara Liskov: I also discovered, though, that they were interested in type hierarchy and that I thought was really interesting.

278
00:46:15.780 --> 00:46:26.580
Barbara Liskov: And it was clear from reading the papers that they hadn't any idea what it ought to me because I read a paper that talked about stacks and hues.

279
00:46:27.240 --> 00:46:40.290
Barbara Liskov: And this paper said that each was a subtype of the other and clearly what they were saying was that they had the same methods with the same arguments and they weren't at all concerned about their behavior.

280
00:46:41.070 --> 00:46:54.300
Barbara Liskov: But it's obvious that if you have a program that expects a stack and you give it a cue isn't going to work because the data, the object that you pass to it doesn't do what it expects

281
00:46:54.780 --> 00:47:13.980
Barbara Liskov: So I gave a keynote at oops law in which I gave a simple common sense definition of sub typing. I said objects of some type, should behave like those of super types. If used to be a super type methods and I mean the behavior.

282
00:47:16.260 --> 00:47:30.810
Barbara Liskov: Was the point I was making clearly you need the syntax to work out. Right. But the behavior is very important. And another thing that this sentence says is that you don't worry about the entire behavior. You only worry about what you can observe. If you think you're observing a super tight.

283
00:47:32.070 --> 00:47:47.700
Barbara Liskov: Okay, so I said that in a newspaper on the newsletter keynote. I did. Subsequently, write a paper about it, which I have a reference for a little bit later. This became a big deal. I wasn't paying much attention. And then one day in

284

00:47:49.020 --> 00:48:01.800
Barbara Liskov: The early 90s. I think I got a an email from somebody saying, can you tell me if this is a correct interpretation of the list car substitution principle. So this name had appeared

285
00:48:02.550 --> 00:48:21.750
Barbara Liskov: And I all of a sudden saw that there was this huge chatter going on on the internet about this principle. What did it mean what's the exact definition and so forth, and subsequently in the 90s Jeanette wing who, as you know, used to be a director of size.

286
00:48:22.770 --> 00:48:30.510
Barbara Liskov: suggested to me that really would be good idea if we pin down what it really meant as opposed to just relying on this very loose.

287
00:48:31.590 --> 00:48:40.410
Barbara Liskov: Common Sense interpretation. And so what I have here is, I have the original paper, which I don't think is a very good definition of what

288
00:48:40.890 --> 00:48:57.090
Barbara Liskov: What behavioral subtype means. And then the paper with Jeanette, which really comes down to describing what it really is. Okay. So that's really what I wanted to talk to you about. I just wanted to tell you a funny story.

289
00:48:58.710 --> 00:49:06.060
Barbara Liskov: When I got the Turing Award in 2010 nine, whatever it was.

290
00:49:07.650 --> 00:49:11.760
Barbara Liskov: Whatever it was, it was not the year the, the word officially was formed anyway.

291
00:49:13.650 --> 00:49:17.670
Barbara Liskov: What is the things that was happening was my husband was on the internet every day.

292
00:49:18.690 --> 00:49:26.460
Barbara Liskov: Looking at all the chatter. And as you know, the internet as we unfortunately know today is not necessarily a very nice place.

293
00:49:27.750 --> 00:49:43.260
Barbara Liskov: And there were comments and some were nice and some are not so nice. This one that I'm going to tell you about was not intended as a nice comment the comment said basically, what did she get the award for we already know this anyway.

294
00:49:44.340 --> 00:49:57.720
Barbara Liskov: But honestly, this was a huge compliment because we didn't know this anyway. In fact, I discovered to my amazement. When I got the award that many of my graduate students didn't know it.

295
00:49:58.050 --> 00:50:04.590
Barbara Liskov: They did not realize that there was a time before data abstraction. They thought it was always there. So,

296
00:50:06.030 --> 00:50:18.510
Barbara Liskov: It's lovely to think that something I invented coupled with the work on small talk and object oriented language led us to what we know today. And that's really the end of my talk. So thank you very much.

297
00:50:21.510 --> 00:50:24.900
Gurdip Singh: Thank you, Barbara Thank you for the great presentation.

298
00:50:26.370 --> 00:50:31.620
Gurdip Singh: For your perspective and your reflection on on

299
00:50:33.090 --> 00:50:39.000
Gurdip Singh: Abstraction and methodology and taking us through this journey through time on to time on how

300
00:50:41.130 --> 00:50:48.780
Gurdip Singh: You know how modularity and abstraction sort of incorporated into the languages and time has become common practice today.

301
00:50:49.440 --> 00:51:14.880
Gurdip Singh: So, so thank you once again for the great presentation. So now we have the floor, open for Q AMP. A. So if you have questions, please go through the Q AMP a tab and write down the equations there and and i will then read through through them. So while we were waiting for those

302
00:51:16.020 --> 00:51:21.030
Gurdip Singh: Questions, of course, those who are panelists Margaret and all you you could

303
00:51:22.260 --> 00:51:25.080
Gurdip Singh: unmute yourself and ask questions as well.

304
00:51:27.660 --> 00:51:45.330
Gurdip Singh: So let me ask a question by while others are putting the equation through. So could you sort of also maybe give your thoughts on how abstraction modularity is being incorporated in the undergraduate curriculum right now and how the

305
00:51:47.970 --> 00:51:50.040
Gurdip Singh: current generation, the next generation of

306
00:51:51.540 --> 00:51:57.750
Gurdip Singh: Students are being trained in this you know how it has evolved over. And what do you think your, your

307
00:51:59.850 --> 00:52:00.660
Gurdip Singh: Vision would be

308
00:52:02.010 --> 00:52:15.660
Barbara Liskov: So I can't talk too much about this because I officially retired in 2014 I haven't been teaching since then I've still been doing research with various colleagues, um,

309
00:52:16.800 --> 00:52:31.830
Barbara Liskov: I certainly can tell you that when I give this talk in to people in industry, I hear from them that their programmers don't know enough about modular design and so they have a problem.

310
00:52:33.480 --> 00:52:47.040
Barbara Liskov: The modules are not necessarily what they ought to be people are violating the rules encapsulation is extremely important. If there isn't something that protects your modules from us from the outside the whole thing falls apart.

311
00:52:47.910 --> 00:52:58.050
Barbara Liskov: Modern programming languages, unfortunately, don't. Many of them do not protect this the way that they ought to do. And so they don't have that helped

312
00:52:58.680 --> 00:53:04.650
Barbara Liskov: And it's always true in a big project with the weakest programmers are the ones that break the system.

313
00:53:05.250 --> 00:53:14.850
Barbara Liskov: So it's unfortunate that we don't do a better job of enforcing encapsulation, because that would be a big help, but I also think it shows that

314
00:53:15.330 --> 00:53:24.690
Barbara Liskov: The students are not being taught properly and i don't know i can tell you at MIT, that the course that I developed six 170

315
00:53:25.410 --> 00:53:45.780
Barbara Liskov: Is still there. It's sort of morphed a few times, but we are still teaching our students about program methodology and modularity and so forth. But I haven't looked at those courses lately, so I don't know. And of course, the world is changing with AI. And it's very interesting.

316
00:53:47.250 --> 00:53:52.200
Barbara Liskov: I mean, the world is changing in many ways. One thing that's been great is to see how

317
00:53:53.220 --> 00:53:55.320
Barbara Liskov: Program verification has come along.

318

00:53:57.660 --> 00:54:01.620
Barbara Liskov: I used to think that it would never be possible to verify anything

319
00:54:02.910 --> 00:54:08.280
Barbara Liskov: You know, except informally, and I always taught my students about informal arguments.

320
00:54:09.720 --> 00:54:26.250
Barbara Liskov: But in some of the later work I did like the work on Byzantine fault tolerance boy would I like to have a formal verification of the core of that system because you can break the core of the system. The whole thing falls apart. And yet, you know, important code is based on that.

321
00:54:28.200 --> 00:54:41.340
Barbara Liskov: Anyway, I don't think that our educational system is in such great shape because somehow I think our students come out and they don't have a good enough understanding of of modularity encapsulation and stuff like that but job.

322
00:54:42.690 --> 00:54:46.440
Barbara Liskov: You know, and I don't have a very broad knowledge and what happens at other universities anyway.

323
00:54:48.900 --> 00:54:49.620
Gurdip Singh: Thank you. Thank you.

324
00:54:50.970 --> 00:54:52.710
Gurdip Singh: So next question here from

325
00:54:53.820 --> 00:54:54.420
Gurdip Singh: Edgar in

326
00:54:55.590 --> 00:55:03.390
Gurdip Singh: Holland important do you think inheritance is now, you know, others may think that it's important has been overstated.

327
00:55:04.800 --> 00:55:17.610
Barbara Liskov: Well, I've never been a fan of heritage. So I've never paid much attention to it. I think it has certainly muddied the water in some ways. I mean, if you think about what happened in Java.

328
00:55:18.630 --> 00:55:21.930
Barbara Liskov: Their solution to the inheritance was to try and use higher

329
00:55:23.220 --> 00:55:33.150
Barbara Liskov: To polymorphous so when Java first came out, I mean, I was delighted see come out. It was the first mainstream language that really had data abstraction in it.

330
00:55:34.500 --> 00:55:42.810
Barbara Liskov: And Andrew Myers was my student at the time. And we tried to get them to put polymorphous and into the language then

331
00:55:43.740 --> 00:55:53.160
Barbara Liskov: And it didn't happen. And there was a mess, you know, and they were trying to do it through inheritance which really doesn't work. And even today, it's kind of a mess and

332
00:55:53.730 --> 00:56:07.440
Barbara Liskov: I don't know that anybody has gotten this whole thing straightened out Andrew Myers is still interested in this question and I don't go to programming language conferences, nor pay much attention to what's going on at them. Um,

333
00:56:09.120 --> 00:56:17.280
Barbara Liskov: So let me put it this way. I really haven't changed my mind. But that doesn't mean but I know Andrew thanks entire inheritances important so

334
00:56:19.050 --> 00:56:27.960
Barbara Liskov: I don't know. You know, the level. I work at today is what one of the things I gave up was programming, I

335
00:56:28.890 --> 00:56:38.850
Barbara Liskov: You know you have a limited amount of time that you can spend on stuff. And so I have chosen to focus on design and

336
00:56:39.600 --> 00:56:50.640
Barbara Liskov: Higher level stuff abstraction, you know, the ideas that sort of put things together. I do tend to work right down to the system level where I think about how things really work.

337
00:56:51.150 --> 00:57:03.090
Barbara Liskov: But the next step down of there's the code, you have to debug it and so forth. I stopped doing that it is extremely time consuming. It's also, of course, lots of fun, but I stopped doing that, um,

338
00:57:05.130 --> 00:57:06.840
Barbara Liskov: I don't know where we started with this, but

339
00:57:09.030 --> 00:57:15.210
Barbara Liskov: Anyway, I think, what I'm saying is, you know, I'm out there in the trenches and so it's a little hard for me to answer that question.

340
00:57:18.270 --> 00:57:27.090
Gurdip Singh: Next question is my coke Bradley's which language today represents your best thoughts and wishes for software developer

341
00:57:28.230 --> 00:57:31.770
Barbara Liskov: OK. So again, this is not something I'm paying attention to.

342
00:57:34.530 --> 00:57:41.490
Barbara Liskov: So it's really hard and I really have not been looking at modern programming languages. I am a little familiar with Python but that's

343
00:57:42.690 --> 00:57:51.630
Barbara Liskov: My knowledge. There's several years old now. I was sorry to see that it didn't have encapsulation. I think the idea of

344
00:57:52.230 --> 00:58:02.610
Barbara Liskov: You know, dynamic static type checking is a good idea. I think, you know, or maybe just separating the idea of what you write. And what you read. So

345
00:58:03.480 --> 00:58:11.670
Barbara Liskov: This is something I often say early in the talk I at the time that I started working on probably methodology. People were very focused on

346
00:58:12.510 --> 00:58:21.810
Barbara Liskov: Making it faster to write code. And in fact, there was even was the name of that language. The one where you could write the one liners.

347
00:58:22.170 --> 00:58:30.630
Barbara Liskov: So you could write these inscrutable little programs in a very few symbols and this was considered a wonderful achievement. But the truth is that

348
00:58:31.350 --> 00:58:35.550
Barbara Liskov: Readability is much more important than right ability you know code is written.

349
00:58:36.240 --> 00:58:45.540
Barbara Liskov: Then you have to read it. Other people have to read it, you know, five years down the line, somebody who isn't even connected to you at all is trying to read it. Um,

350
00:58:46.290 --> 00:58:58.170
Barbara Liskov: So a way to sort of bridge that gap is to make it easier to write and have the support system fill in the details. So, for example, people don't like to write down data types.

351
00:58:58.890 --> 00:59:10.590
Barbara Liskov: But it's easy for the, you know, for the runtime system to fill that kind of detail in. I think that's a great direction. I'm not following these conferences, so I don't really know what the current state of the art is

352
00:59:11.160 --> 00:59:18.300
Barbara Liskov: But I think that is a promising direction and is helpful. And as I said, of course, cancellation.

353
00:59:20.550 --> 00:59:28.050
Barbara Liskov: Yeah. And, and I get one more, one more thing I didn't do a lot of working concurrency recently and

354
00:59:29.130 --> 00:59:44.370
Barbara Liskov: Of course encapsulation is even more important there. If you don't, if your module is not in charge of how the concurrency is working inside your highly concurrent data type, you know, all hope is over. I mean, it was already over without concurrency, but it's even worse now.

355
00:59:45.570 --> 00:59:51.630
Barbara Liskov: And furthermore, you don't want a programming language deciding what that concurrency is because

356
00:59:53.040 --> 01:00:08.400
Barbara Liskov: As you know about the time I did the work on clue. All we knew about was locks and then after while there was optimistic concurrency control and now there's all this RC you going stuff going on, you know, read copy update stuff where

357
01:00:09.570 --> 01:00:17.700
Barbara Liskov: You don't even, you know, you don't have any locks and you know this stuff is all perfectly manageable. If you just have an encapsulated module in which the

358
01:00:18.240 --> 01:00:28.050
Barbara Liskov: Complicated, you know, rules are being carried out, so it's encapsulation matters, having a programming language that prescribes your concurrency control mechanism, just not

359
01:00:30.060 --> 01:00:33.270
Gurdip Singh: there yet. So the next question is,

360
01:00:34.290 --> 01:00:34.830
Gurdip Singh: From

361
01:00:35.850 --> 01:00:45.870
Gurdip Singh: malgorzata sketches. She thanks you for the great talk. And the question. She has is the use of global variables as bad as the go to State and

362
01:00:46.440 --> 01:00:49.140
Barbara Liskov: I know. Can you repeat that please because I didn't quite get

363
01:00:49.230 --> 01:00:54.690
Gurdip Singh: It is the use of global variables as bad as the go to state.

364
01:00:56.370 --> 01:01:04.230
Barbara Liskov: Yes, it is as bad. Now notice that, I mean, in fact, it's interesting to look back and think about algos 60

365
01:01:04.890 --> 01:01:15.540
Barbara Liskov: Okay so algos 60, you know, it had this great innovation. It had these inner blocks and the variables inside the inner block could not be accessed from the outer block. So that was good.

366
01:01:16.110 --> 01:01:23.700
Barbara Liskov: But on the other hand, it was all set up so that you would communicate through the variables on the hour block, you don't want to do that at all. You really want

367
01:01:24.450 --> 01:01:32.880
Barbara Liskov: You have your modules, they're independent, but notice that there is a kind of global variable and all of our programs and that is the file system.

368
01:01:33.420 --> 01:01:45.120
Barbara Liskov: You know the database their global the global variables are still with us. I think they're using a much more disciplined fashion now than they once were. But they, we did not get rid of them. And I don't think you can

369
01:01:51.060 --> 01:02:06.210
Gurdip Singh: Phillip show on past. So this is a question which you touched upon briefly earlier to the question I asked, it says, how do you approach teaching of programming modularity. What kind of examples or two. Would you use

370
01:02:07.230 --> 01:02:09.450
Gurdip Singh: Could you give us some idea.

371
01:02:10.230 --> 01:02:12.570
Barbara Liskov: Well, I can tell you what I used to do.

372
01:02:13.950 --> 01:02:20.940
Barbara Liskov: It actually probably the person you who might have the best ideas. So I have a couple of colleagues who are basically teaching

373
01:02:22.020 --> 01:02:25.590
Barbara Liskov: The kinds of stuff that was developed in six 170 so john doe tag.

374
01:02:26.610 --> 01:02:43.260
Barbara Liskov: Has an introductory course and trainee given to us. Also, and they use examples, and I don't know what those examples are but they're very different what I used to use. I used to use quick index as an example. And it was, you know, a sequential program, but

375

01:02:44.430 --> 01:02:56.460
Barbara Liskov: And I think it was kind of boring for the students. You know, so you would use an example. I mean, first of all, I taught them about the methodology, the one I mentioned earlier about inventing abstractions and imagine you had an abstract machine.

376
01:02:56.910 --> 01:03:09.120
Barbara Liskov: And then I would illustrate it by doing a design, starting from a problem statement and going through now of course what I've been doing in the last 30 years 40 years

377
01:03:10.020 --> 01:03:18.960
Barbara Liskov: Is thinking about systems which are not sequential systems. And so, you know, they're you're thinking more in terms of majors, you start with your major subsystems.

378
01:03:19.500 --> 01:03:28.440
Barbara Liskov: And I might work more from that level today if I was teaching that course I would think about. It's an interesting question. You know, you have to walk before you can run

379
01:03:29.910 --> 01:03:36.060
Barbara Liskov: And I tend to think in terms of distributed systems and highly concurrent systems.

380
01:03:37.650 --> 01:03:38.190
Barbara Liskov: So,

381
01:03:41.220 --> 01:03:42.690
Barbara Liskov: I think it's a great question.

382
01:03:44.190 --> 01:03:51.660
Barbara Liskov: John has a book and I bet you he's got some interesting examples in that book that he uses for teaching. So that might be good place to look.

383
01:03:53.130 --> 01:04:00.390
Barbara Liskov: Quick inject quick index is a very good example, but of course I haven't actually taught that course since the 90s. So it's been a while.

384
01:04:03.840 --> 01:04:05.850
Gurdip Singh: Next question is from

385
01:04:07.140 --> 01:04:27.540
Gurdip Singh: when when when from University of Georgia. So thank you for the wonderful insightful task talk and two questions. So the first one is, do you think abstractions also removes some potential optimization opportunities that can be exploited by components.

386
01:04:28.680 --> 01:04:44.640

Barbara Liskov: Of abstraction is absolutely not in the way of what the compiler can do, in fact, a very powerful technique. Well, remember in line substitution. So, you know, that's a simple optimization technique where the compiler.

387
01:04:45.390 --> 01:04:58.620
Barbara Liskov: Exposes to itself. The code of the procedure and then does all sorts of manipulations of the code so that avoid the cost of the calls and so forth and so on. And you can get rid of variables and all sorts of stuff and then

388
01:04:59.070 --> 01:05:08.010
Barbara Liskov: If you change your idea you re implement procedure, no problem. You just re compile the deal. So in fact, I think it's quite the opposite. You know abstraction.

389
01:05:08.430 --> 01:05:14.820
Barbara Liskov: Is something that you think about, but you don't actually reason in terms of the code that runs on the machine.

390
01:05:15.450 --> 01:05:23.940
Barbara Liskov: Although of course if the compiler is doing stupid things you may run into a little problem. You know, like all those problems in the C plus positive either where it doesn't

391
01:05:24.330 --> 01:05:37.620
Barbara Liskov: It gets a little bit over ambitious about what it's doing with your code. So the compiler had better be doing his job properly, but really the idea that you work in an abstract level and then the compiler comes in and does all this manipulation. That's a great use of abstraction.

392
01:05:39.450 --> 01:05:55.320
Gurdip Singh: Yeah. Yeah, I agree. Yeah, I'm for the second question is big merger. Well, it says, given the recent advances in machine learning. Do you think computers will eventually be able to program themselves.

393
01:05:56.490 --> 01:06:01.440
Barbara Liskov: Yeah, isn't that an interesting question. Yeah, let me put it this way. Well, first of all,

394
01:06:04.500 --> 01:06:11.730
Barbara Liskov: You know, there's a lot of issues with what's going on in our world today and quite a few of them are due to machine learning.

395
01:06:12.810 --> 01:06:21.330
Barbara Liskov: So I think we should be putting our energy into trying to, you know, do whatever technologically, we can do about fake news and

396
01:06:22.350 --> 01:06:28.680
Barbara Liskov: I mean, just think of the problems we have to face of machine learning gives you

397

01:06:29.880 --> 01:06:33.870
Barbara Liskov: 95% correctness. I mean it.

398
01:06:34.890 --> 01:06:50.310
Barbara Liskov: Today, it doesn't give you 100% programming is very precise. So I don't think we're ready for prime time here, although I know I have colleagues who think this is cute and

399
01:06:51.840 --> 01:06:59.670
Barbara Liskov: You know, somehow you get an almost correct solution. I mean, I suppose, if you're in an application we're almost correct is good enough, it might work.

400
01:07:00.540 --> 01:07:13.140
Barbara Liskov: But that seems like the big deal to me. The fact that we don't quite get all the way there. And we want to get all the way there so that the code really works. And that's a big problem.

401
01:07:14.880 --> 01:07:23.670
Barbara Liskov: But, you know, the issue about what's going to happen to software. What's going to happen to programmers.

402
01:07:25.290 --> 01:07:31.440
Barbara Liskov: You know, our world is changing due to computer science and

403
01:07:33.000 --> 01:07:34.170
Barbara Liskov: Who knows where we're going.

404
01:07:39.570 --> 01:07:42.900
Gurdip Singh: So it's to have a show from

405
01:07:43.980 --> 01:07:44.730
Gurdip Singh: Terry

406
01:07:45.750 --> 01:07:55.320
Gurdip Singh: lakin down live in June. So this is, this is not a question actually. But would be interested in your talked about this is the

407
01:07:55.860 --> 01:08:13.800
Gurdip Singh: Modularity was developed about the same time in linguistics. In the 1970s, for the design of grammatical systems and Jerry photo famously promoted the idea for the design of mind in the modularity of mind in 1975

408
01:08:15.090 --> 01:08:22.170
Gurdip Singh: My impression is that these ideas were not pursued with the same rigor, as they were in computer science.

409
01:08:23.460 --> 01:08:23.850
Gurdip Singh: So,

410
01:08:25.350 --> 01:08:34.590
Gurdip Singh: Yeah, I mean, I see that you know there was no overlap in terms of languages design and all between linguistics and and and computer science. So

411
01:08:36.570 --> 01:08:42.060
Barbara Liskov: Yeah I you know I That's way beyond my level of expertise. It was just striking me

412
01:08:44.100 --> 01:08:47.280
Barbara Liskov: You know, when we speak. We can be ambiguous.

413
01:08:48.510 --> 01:08:51.600
Barbara Liskov: When we speak to a computer. We can't be ambiguous.

414
01:08:54.450 --> 01:09:00.360
Barbara Liskov: I'm not sure that there is that much of a connection here, but I don't know its way out of my

415
01:09:04.860 --> 01:09:17.520
Gurdip Singh: School into the equation. This another one on how we teach our students to design mode. So how do we teach our students to design more effectively. So I know you've already talked about it and you may have

416
01:09:17.940 --> 01:09:31.710
Barbara Liskov: Well, I had the students work in teams. So I started them off working on very small projects where they still had to design, but it was small, they can do it as a single person.

417
01:09:32.550 --> 01:09:40.530
Barbara Liskov: And then later in the course. I had them work on teams on a larger project where they really had to sub divide and come up with a

418
01:09:42.000 --> 01:09:46.470
Barbara Liskov: System that worked. And that was, I would say only

419
01:09:47.670 --> 01:09:56.280
Barbara Liskov: Hardly successful and the problems were not well first of all, I should say. I actually don't believe you can teach design.

420
01:09:56.910 --> 01:10:11.640
Barbara Liskov: So, I believe, maybe you can teach small scale, you can teach the principles of design, but not everybody has the ability to design it it's it's a it's a skill. It's a, it's almost artistic, you know, because the design.

421
01:10:13.200 --> 01:10:27.120
Barbara Liskov: A good design has kind of elegance to it. So all I could teach was the principal design. I could try to get across. You know, the modularity work. So if you came up with a good design, it would be better than if you didn't come up with a good design.

422
01:10:28.650 --> 01:10:44.640
Barbara Liskov: The, the problem with that team project also had always had to do with people. It didn't have to do with, you know, and maybe these problems exist in industry to but teams didn't always work. And it was very hard to

423
01:10:45.840 --> 01:10:54.090
Barbara Liskov: Monitor this and to help out the students, you know, you have teams where they'd be one really strong programmer and they would just take over everything, and nobody got to do much.

424
01:10:54.600 --> 01:11:03.630
Barbara Liskov: Or you could have. There were lots of issues with the women you know they had teams where there was a woman on the team, and she was sort of sideline and

425
01:11:04.650 --> 01:11:13.470
Barbara Liskov: So there were lots and lots of issues with running teams, but since large projects and modularity, have to do with teams and something you have to go there because

426
01:11:14.430 --> 01:11:20.970
Barbara Liskov: Throttle you get something big, you know, a small program you write yourself. Well, you probably made more notice that it was a problem. So,

427
01:11:23.010 --> 01:11:29.100
Gurdip Singh: Yeah, I think. Yeah. Modularity maps nicely our teams map nicely into modularity and they go ahead

428
01:11:31.080 --> 01:11:41.910
Gurdip Singh: So car going on the coupon offers a question, what are your perspective on how to address the security caps that software code seem to have overall

429
01:11:44.790 --> 01:11:45.510
Barbara Liskov: Yeah.

430
01:11:46.740 --> 01:11:50.220
Barbara Liskov: Are you thinking about the recent hacking into the government computers.

431
01:11:54.330 --> 01:11:59.580
Barbara Liskov: Yeah you know this is hard. My people who aren't my friends who are

432
01:12:00.870 --> 01:12:07.200

Barbara Liskov: Working in this area, you know, they think about it as a game, you know you appeal the security of your system.

433
01:12:08.880 --> 01:12:20.550
Barbara Liskov: But the hackers are very are very motivated to break it, it would be if we use better programming languages that might help you know the lot of the

434
01:12:21.810 --> 01:12:33.690
Barbara Liskov: Ways, they did the vulnerabilities are often do to problems in the software that could have been avoided if we'd had a better programming language to begin with. It would be good. Maybe if we didn't have so much legacy code.

435
01:12:36.990 --> 01:12:38.040
Barbara Liskov: But of course,

436
01:12:40.170 --> 01:12:48.930
Barbara Liskov: The real problems are human interface problems, you know, how do we get people to not click on a link that they should not click on

437
01:12:50.610 --> 01:12:55.230
Barbara Liskov: So in answer to your question. You can see I don't have any solutions.

438
01:12:56.220 --> 01:13:01.890
Gurdip Singh: Yeah, and it's also goes back into how do you teach them to write more effectively and

439
01:13:03.000 --> 01:13:19.680
Barbara Liskov: Yeah, but yeah. But I think what I'm saying is, even if you had a perfect system if there were any way that clicking on a bad link is going to be a problem. And you know, I mean, you can't help with that being a problem, you know, so people who have worked on.

440
01:13:21.420 --> 01:13:29.010
Barbara Liskov: User Interfaces and trying to figure out ways to design better interfaces and and nobody has a good solution, you know, the

441
01:13:29.670 --> 01:13:43.500
Barbara Liskov: You know, Pat. People don't like to use passwords that are complicated captures on they'll be useless because machine learning will be able to solve them anyway and and they're annoying so

442
01:13:44.550 --> 01:13:48.270
Barbara Liskov: I think, you know, we, yes, we should get our act together in the in the

443
01:13:48.750 --> 01:13:56.820
Barbara Liskov: In the systems and build a software that actually works. But there's even if we did that. There's a whole

other area of problems, having to do with user interfaces.

444
01:13:58.260 --> 01:14:00.330
Barbara Liskov: And so it's a very hard problem.

445
01:14:02.820 --> 01:14:03.300
Gurdip Singh: Who

446
01:14:04.950 --> 01:14:17.370
Gurdip Singh: Knows, we should move from Johns and he asks, what would be the most disruptive phenomena in programming or computation in general. Since 2004

447
01:14:18.750 --> 01:14:20.520
Gurdip Singh: Your retirement, in your opinion.

448
01:14:20.730 --> 01:14:31.350
Barbara Liskov: Oh, well, what's going on the machine language because they really mean machine learning. Absolutely. So when I was at Stanford. I actually was interested in machine learning.

449
01:14:32.040 --> 01:14:47.910
Barbara Liskov: But machine learning in those days was you write a program that thinks like a person. And that really wasn't very effective. And one of the reasons I left AI was because I thought this is never going to work, or it's too hard for me or something like that.

450
01:14:49.680 --> 01:14:58.050
Barbara Liskov: But, you know, things are totally different now. And what you can do with machine learning is quite amazing. So I mean, things have really changed. There's no doubt about it.

451
01:15:02.760 --> 01:15:16.560
Gurdip Singh: So I see no more questions. So I have one question. So I know that if you talked about, you know, in terms of looking at program. We go to statements and all the abstractions made it difficult to

452
01:15:17.700 --> 01:15:33.630
Gurdip Singh: Orange or during CES 70s, 80s, the hardware itself a sequential where it was easy to sort of see how the program execute and debug. So I'm just curious about your thoughts on how the advances in hardware have

453
01:15:34.800 --> 01:15:35.250
Barbara Liskov: So I

454
01:15:37.380 --> 01:15:42.270
Gurdip Singh: The reasoning about them, or even how you write and all the

455

01:15:43.500 --> 01:15:45.660
Gurdip Singh: abstractions that you may have. So

456
01:15:46.740 --> 01:15:54.960
Barbara Liskov: So I'm I don't quite understand your what your are you asking me about how computers run today versus how they used to run

457
01:15:55.200 --> 01:16:03.060
Gurdip Singh: Know, or how the advances in the hardware, the multi processing systems that we have and all impacted abstractions.

458
01:16:04.350 --> 01:16:21.990
Barbara Liskov: I don't think they've impacted abstractions, or at least in my experience, I mean, I, I've spent a lot of time in the last five or six years working on multi core computers. So I'm talking about highly concurrent programs.

459
01:16:23.340 --> 01:16:37.140
Barbara Liskov: abstractions are our salvation there. And you know what I said earlier about modularity and encapsulation being the, you know, the thing that really matters. Now I don't have experience.

460
01:16:38.130 --> 01:16:44.010
Barbara Liskov: With a lot of the other stuff that's been going on. So I don't. I really can't say anything about that.

461
01:16:44.460 --> 01:17:02.280
Barbara Liskov: But I can tell you that, as far as distributed computing is concerned, as far as highly concurrent programs are concerned abstraction. If anything is more important than that it isn't a sequential simple sequential world that I was working in. When I first started working in this area.

462
01:17:06.210 --> 01:17:06.930
Gurdip Singh: So,

463
01:17:07.980 --> 01:17:10.350
Gurdip Singh: See no more questions. So again,

464
01:17:11.850 --> 01:17:16.320
Gurdip Singh: Thank you, Barbara once again for the great presentation.

465
01:17:17.460 --> 01:17:39.930
Gurdip Singh: Really be thoroughly enjoyed it, and I'm sure all of our audience has also enjoyed the presentation. So thank you once again. So I just also want to remind everyone that at four o'clock today, at least for all of the NSF offers on the zoom call that we have will have office hours.

466
01:17:40.980 --> 01:17:46.590
Gurdip Singh: With Barbara. So please do join us at four o'clock today, so thank you once again.

467
01:17:47.010 --> 01:17:49.230
Barbara Liskov: Thank you. You're welcome. Bye bye.

468
01:17:49.710 --> 01:17:50.040
Bye bye.