Barbara Liskov on Programming, Career, and the Future

Barbara Liskov

Barbara Liskov (http://www.pmg.csail.mit.edu/~liskov) is the Ford Professor of Engineering in MIT's http://www.eecs.mit.edu/ Department of Electrical Engineering and Computer Science and head of its Programming Methodology Group. She has led several major projects, including the design and implementation of CLU, the first language to support data abstraction; Argus, the first high-level language to support implementation of distributed programs; and Thor, an object-oriented database system that provides transactional access to persistent, highly available objects in wide-scale distributed environments. She recently received the IEEE John von Neumann Medal for 2004 for her "fundamental contributions to programming languages, programming methodology, and distributed systems."

*IEEE Distributed Systems Online* editor Dejan S. Milojicic contacted Liskov to discuss her career, interests, choices, and advice to young researchers just entering the field.
Dejan Milojicic: Barbara, congratulations on the award that put you beside such renowned researchers as Alfred Aho, Butler Lampson, John Hennessy and David Patterson, Maurice Wilkes, and Gordon Bell.

You've had an amazing research career. You've worked successfully in many areas—operating systems, programming languages, databases, fault tolerance, and distributed systems. How do you identify new areas of research? Are you driven by industry needs or by academic intuition?

Barbara Liskov: I'm driven more by intuition than by industry needs. My interest is in tackling problems that need to be solved in the sense that the lack of a solution is a barrier to progress. Thus my early work on programming languages was concerned with how to organize programs so as to make it easier to develop them and have them be correct. This work was done in the 1970s at a time when very little was understood about program structure (this was before the invention of object-oriented programming). My work on distributed systems in the '80s was concerned with how to organize and think about distributed programs. At that point, people were just beginning to think about how to build these programs and didn't understand much about their structure.

What ideas permeate all the areas you've worked on? What must researchers be aware of no matter what they're working on—for example, hands-on implementation, clean architecture, careful design, security, and scalability?

I've changed research areas over time. The common thread has been my interest in working on problems whose solutions were needed at the time, as I already mentioned. Today, I'm primarily interested in large-scale distributed systems that continue to perform correctly even in the presence of failures and malicious attacks.

I believe the most important attribute in carrying out research is to thoroughly analyze and understand the problem and why your approach to solving it is the right one.

In my research area of software systems, it's also very important to reduce ideas to practice. This means that the solutions I invent need to be "sufficient" to solve the problem; they should be as simple as possible, but the system has to really run, and it has to run with good enough performance. There is often a trade-off between simplicity and performance, and one of the key issues in doing design is to resolve it in the right way.
Then it's always necessary to implement the solutions and do experiments to see how they work in practice. Here careful design and a clean architecture are very important.

**Have you been able to transfer your experience from one area to another? For example, from Venus to CLU, to Argus, to Thor, and so on?**

There's a very straightforward progression among these projects. For example, I designed Venus using a technique I invented to make the implementation highly modular; I developed this technique as a way of keeping the complexity of the software under control. When Venus was finished, I abstracted from its structure to come up with a general approach to modularizing programs. I called this approach the *multi-operation module*. While thinking about how to explain this idea to others, I realized that it was possible to think about these modules as data types, in which the objects belonging to the type had operations that could be used to interact with them. Then I decided to design a programming language, CLU, to explore this idea and pin down all its details.

**If you had a chance to repeat some of your work from the past, what would you do differently? Can you extrapolate this question to examples of other big projects and research, in other words: what if...?**

There were points in my career where I had to choose whether to go on to a new research area or spend time solidifying and selling earlier results. I always chose to go on, and I don't regret that. But it would have been nice to be able to do both.

**Looking back, what are you most proud of as a researcher, developer, and technologist? What would you do differently if you had the opportunity?**

I'm probably most proud of my work on CLU and data abstraction, my work on Argus and the structure of distributed systems, and my work on replication techniques. I've been very lucky to have found a field that suited my abilities and where I was able to see the right problems to work on. I was able to make contributions and also have a lot of fun.

**Identifying the right problems to address is usually the hardest part of research. What do you recommend to young researchers?**

My main advice is to avoid incremental work. You should be aware of what others have done and take advantage of it. But rather than thinking of small ways to improve it, try to
think of how to solve the problem differently, or how to apply the technique to a different area, or how to abstract from what has been done to come up with something that's broadly applicable.

**Are you the primary instigator of the new domains that you and your students research, or do new students' interests drive your work?**

It can work either way. I usually have some topics that I'm particularly interested in at any point in time. I like to work with students on research related to these topics. This provides a context within which students can do their own research. Sometimes students are able to identify their own topic. Other times they pick up a topic of interest to me, but they provide their own solution.

**Why haven't we influenced industry sufficiently? For example, what programming techniques would have made a difference today but haven't caught on?**

Actually, I think research does have an impact on industry. The problem is that the time lag can be quite long.

**How would you divide the roles of researchers (academia and labs) and developers (industry, startups, open source)? Where does one stop, and where should the other take it over?**

Researchers should propose novel ways of solving problems. They should also reduce them to practice, and often this involves building prototypes. This way they can provide a proof of concept. After that, the technology needs to move to industry, which is much better equipped to build industrial-strength software.

**Who do you see as key drivers and catalysts in systems research today? Traditional computing companies? Academia? Small companies and startups? Open source communities?**

I think the key drivers are research and startups. Researchers often decide to see if they can have impact by starting a company to produce a product based on their ideas. This doesn't seem to work so well via established companies.

**Which accomplishments in computing science have made the most difference, in
your opinion? Where is the most need for future breakthroughs?

There have been many such accomplishments, and there will be many more. But I believe the central unsolved problem is our inability to build programs that work correctly. A breakthrough here would be very nice (but I'm not very optimistic).

Every other year, there's a new fashion. A few years ago, it was P2P; recently, it's been grid computing. Do you think this is a promising technology or only the most recent fashion?

There definitely is a kind of marketing going on. But in fact, both P2P and grid cover research that is addressing key problems, such as how to harness the power of lots of computers to carry out a computation that would otherwise take a very long time.

Where do you see computer programming of the future? How are users going to deal with complexity, security, and so on?

Programming's central problem is complexity. Programs are very complicated, and typically no one understands them completely. As a result, they don't work correctly. We've made strides over the years in the size and ambition of the systems we build: we can tackle much larger problems today. But we still don't know how to make sure that our programs really work. I hope there will be new ideas about how to manage complexity.

As a user, what's your ideal vision of the next computing era? As a researcher, how do you see this vision coming true?

As a user, I'd like to see systems that are easy to use and that protect me from making stupid mistakes, such as releasing private information to unauthorized parties. I believe that research will lead to systems that are better in these respects.

Can you offer advice to researchers at the beginning of their career? What are the recipes and rules for success, if there are any at all? What are the promising areas they should address, and which ones might not lead to fruitful research?

There are no recipes or rules. And young researchers are best at spotting the hot new areas. But here are some things that might help:
● Ask lots of questions. It's good to question the assumptions that others make. This can often lead to insights into better ways of doing things.

● Avoid doing incremental work. Instead, look for things that make a difference.

● And be honest. Think carefully about what you're doing. Consider both pros and cons. The cons are especially important.