Dr. Debasish Jana
Editor, CSI Communications
Vice Chairman, CSI Kolkata Chapter

Exclusive Interview with Prof. Jeffrey David Ullman

Father of database, automata and compilers

(L to R) Debasis Bandyopadhyay, Jeffrey Ullman, Debasish Jana, Sushanta Sinha and Prateep Misra

(L to R) Jeffrey Ullman, Debasish Jana and Sushanta Sinha, a copy of CSIC, August 2011 issue is seen on the table

Jeffrey David Ullman is the Stanford W. Ascherman Professor of Computer Science Emeritus at Stanford University. He is the author or co-author of sixteen books, all setting standards in their respective fields. His popular textbooks include compilers (various editions popularly known as the Dragon Book), theory of computation (also known as the Cinderella book), data structures, and databases. His research interests include data mining, information integration, and electronic education.

Ullman received BS in Engineering Mathematics from Columbia University in 1963 and PhD in Electrical Engineering from Princeton University in 1966. His early job was at Bell Labs He was Professor at Princeton from 1969 to 1979. Since 1979, he has been a Professor at Stanford University.

Ullman's research interests include database theory, data integration, data mining, and education using the information infrastructure. He is one of the founders of the field of database theory, and was the doctoral advisor of an entire generation of students who later became leading database theorists in their own right. He was the Ph.D. advisor of Sergey Brin, one of the co-founders of Google, and served on Google's technical advisory board. He is currently the CEO of Gradiance.

He is a member of the National Academy of Engineering, ACM Fellow, recipient of a Guggenheim Fellowship, the Karl V. Karlstrom Outstanding Educator Award, the SIGMOD Contributions and Innovation Awards, and the Knuth Prize (2000). Ullman is also the co-recipient (with John Hopcroft) of the 2010 IEEE John von Neumann Medal, "for laying the foundations for the fields of automata and language theory and many seminal contributions to theoretical computer science".

Prelude

In September, 2011, Jeffrey D Ullman came to India as part of his periodic tours of India for TCS. During his trip to India, he visited Bangalore, Pune, Delhi and Kolkata. During his visit at Kolkata, this interview was initiated with active help and cooperation from Arpan Pal, Debasis Bandyopadhyay and Sushanta Sinha of TCS Kolkata. Excerpts from his postings at Google Plus on his Kolkata trip: "Friday was Kolkata. If you think of Kolkata (or Calcutta as it was known) through the eyes of Mother Teresa, you think of it as the armpit of the universe. But that is very far from the truth. Compared with most of the major Indian cities, the roads are well paved, with lines demarking lanes and few if any potholes. Traffic was not bad, and the only major jam I was in, on Friday night, seemed to be people going out for entertainment."

Part 1 of 3

On Core Fundamentals of Computing and Future of Education

DJ: *I feel extremely privileged to meet you in person at Kolkata, the City of Joy. At the same time, I feel indebted to the active cooperation of Sushanta Sinha, Arpan Pal and Debasis Bandyopadhyay of TCS to initiate this interview while you were at Kolkata in September, 2011. Very special thanks to you for kindly agreeing to an exclusive interview for Computer Society of India.*

Down memory lane

DJ: *The month of October, 2011 is extremely bad. Two great stalwarts: Steve Paul Jobs and Dennis Ritchie have left us. The world of computing has faced a great loss by these two demises. Please share some great moments with them.*

JDU: I wish I could. I never met Steve Jobs. I was at Bell Labs for a few years when Dennis Ritchie and Ken Thompson were beginning their work on UNIX. Dennis was always a very decent fellow, but alas our interests were very different at that time. And the bad news continued with the death of John McCarthy.

Our approach is to teach the subject through experience, first in instructor-guided team projects, and later in individually chosen research.

DJ: *Yes, that's too bad news to the world of computing. Death is inevitable to everyone, the eternal truth of life. Let me quote Rabindranath Tagore, who sparkled enriched cultural renaissance throughout India through his songs, drama, painting and above all countless literary works, "When my footprints will not be here in this hut, when I shall not sail the boat here in this river, at that time you may not even remember me–still I shall be here, my soul will be here". The work, the inner soul, the footprints don't perish.*

Software Engineering: Teaching and Practice

DJ: *The field of software engineering is very dynamic in nature. Owing to the dynamic nature of the field, there will always room for innovation and improvement. But there is always a gap found in software engineering theory taught in many undergraduate and graduate degree programs versus software engineering practices. On one hand, students feel software engineering is a boring subject, whereas practitioners live on these. And there is a huge gap. What is your perception and suggestion on this?*

At Stanford, we try to educate people who will get a software job right out of college, those who will go for a masters, and those who will eventually get a doctorate and enter research or advanced development.

JDU: At Stanford, we do not have courses called "Software Engineering". Our approach is to teach the subject through experience, first in instructor-guided team projects, and later in individually chosen research. But I am puzzled by the claim that SE is a "boring" subject. There are two kinds of "theory" that I know about in SE. One involves automated program proving or checking or other kinds of automated analysis. In the US we tend to be less enthusiastic about proofs of correctness but quite interested in model checking, static analysis, and other theoretical techniques that have had significant successes. These subjects can be taught in an interesting manner and can be quite instructive. The other kind of theory is more philosophical in nature. I cannot judge the importance of these ideas; I'm sure each of them has its following and some successes to point to. I am less surprised that people find philosophical material boring. CS attracts people who want to do things, not philosophize about them. They'll learn the methodologies they need in the field, when their job requires it.

Education in computing: the scope, focus and coverage

DJ: *Comparing with the early ages when Computer Science course was introduced as a formal discipline in many parts of the world, and today, we have Computer Science and many other variants like Information Technology, Information Science etc. What is your vision of imparting education in computing as needed for today's arena?*

Today's curriculum should be flexible enough that students with interest in the fundamentals (e.g., theory of algorithms) should be able to get that instruction, and yet let those who are looking for a job writing applications concentrate on pragmatics.

JDU: There isn't just one arena. At Stanford, we try to educate people who will get a software job right out of college, those who will go for a masters, and those who will eventually get a doctorate and enter research or advanced development. Thus, we offer many different tracks at all levels, ranging from theory to systems to applications, and allow students to choose with only a small core curriculum. We keep the Computer-Science label for all these programs, but we could have identified some of the lower-level tracks as "Information Technology" or some similar term.

DJ: *Because of service orientation approach in software development, the innovation in computer learning and investment for compilers have become less especially in Indian scenario. Algorithms, data structures, databases, compiler design, fundamental programming paradigms are all extremely*

essential to know, to preach, to teach the underlying concepts and vision. Please talk about coursework and what subjects should be taught today in a CS curriculum.

The theory that has withstood the test of time, such as finite automata, and the basic algorithms ideas are things that every student should learn.

JDU: Compilers were once a major portion of the curriculum. Not only were they the best example of how theory could impact practice, they were the point in the curriculum where we asked students to write a nontrivial piece of software. Today, there is so much else to learn, and compilation is so small a portion of the total picture that the course is optional rather than required at most schools.

At the same time, the software industry has grown to the point where there is a well-established hierarchy of people who code the designs of others, those who do the designs, and those at the top who decide what to design for. It is essential we prepare students to enter at the lower rungs and enable those who are capable to reach the higher rungs. Today's curriculum should be flexible enough that students with interest in the fundamentals (e.g., theory of algorithms) should be able to get that instruction, and yet let those who are looking for a job writing applications concentrate on pragmatics. The approach at Stanford is to allow students to take a number of courses in specific languages or systems, e.g., "writing Android apps" or "Python programming," but not to count these courses toward the units required for a CS major. I think that is a good compromise.

Theoretical knowledge and innovation in practice

DJ: *You once told "theory is obsolete", how does it stand for young learners, even for practitioners in age-old processes?*

JDU: I don't remember ever saying that. There is a lot of theory that is pointless game-playing, but theory is still very important in a number of areas, including cryptography, computational complexity, design of algorithms, and hardware or protocol verification. Fields, such as databases, graphics, and AI, have developed their own specialized theory that strong practitioners need to master. Many believe that the theory has impact in verification of hardware, and even software, especially protocols. Moreover, the theory that has withstood the test of time, such as finite automata, and the basic algorithms ideas are things that every student should learn. These concepts get applied all the time.

I think that people able to handle a little bit of the theory will generally be more adept at software development.

DJ: *In fact, the innovation in computer learning and investment for compilers have become less especially in Indian scenario. Many universities are not offering compiler design courses. In fact, most of the jobs these freshers coming out of colleges see seldom require knowledge in automata, compiler design, graph theory, algorithms and core foundation of computer science. It's very sad, and disheartening but yet true. As such, quality students, quality teachers, quality practitioners are less. Less is more, more is less, we know. But, still, what do you say?*

JDU: I responded to the matter of compilers earlier. I know that CS theory plays a reduced role in education, compared to what it was when I started teaching in the late 1960's. But much of that is because there are so many more pragmatic things worth learning today. But there is still a core of theory, much of it covered by the topics you mentioned in your question, that are worth teaching to everyone. Moreover, I think that people able to handle a little bit of the theory will generally be more adept at software development, even if they don't use the specifics taught in their theory classes, because software creation and mathematics are similar in their demands for rigorous thinking, precision, patience, and a number of other qualities. Thus, a good theoretical curriculum encourages the right kind of people to take up a career in software.

Cloud computing is another example of the triumph of commodity hardware. Instead of specialized "supercomputers," we get more cost-effective parallelism from racks of commodity processors and disks.

Changing focus of computing

DJ: *How do you see the changing trend form of computational shift from centralized to distributed, in-house to cloud computing?*

JDU: I find it unsurprising. As should be apparent, there was a kink in the Moore's law curve about a decade ago, and the consequence is that you can't get more speed out of a single processor. That forced us to go parallel.

Cloud computing is another example of the triumph of commodity hardware. Instead of specialized "supercomputers," we get more cost-effective parallelism from racks of commodity processors and disks. History has many examples where trying to design hardware for specialized applications fails in the face of less efficient (for that problem) commodity hardware. The failures of Lisp machines and word-processors come to mind. I'm sure there are many others.

About Gradiance: creating homework a learning experience

DJ: *Please tell us about Gradiance, system for creating and administering class exercises, in cloud perspective*

JDU: Almost 10 years ago, four of us– Ramana Yerneni, Alan Beck, Murty Valiveti, and I – developed a platform for managing homeworks. Ramana was the principal architect, and Murty, who runs Gautami Software in Hyderabad, contributed the design and implementation.

The key difference between Gradiance and other homework platforms is that our goal is to make homework a learning experience, rather than a hoop for students to jump through. We invented the "root

question," which is a form of question in which there are many right and many wrong answers. Thus, question designers have to be a bit careful how they phrase questions. So instead of asking "compute the join of these two relations," you ask "compute the join of these relations and then, in the list below, identify the tuple that is in the join." In that way, there are many possible correct answers, one for each tuple in the join. We ask the student to solve the problem completely, and then answer a multiple-choice question about the answer. For the join question, we would have them compute the join just as they would in a conventional homework, but then they are given one correct tuple and three incorrect ones. If they pick an incorrect tuple, we offer them a "choice explanation," which in this case would address the reason why their choice was not in the join, as well as general information such as the definition of the join. Students are then allowed to take the same question as many times as they like, until they get it right. To discourage random guessing, we bundle several questions in a group and ask them to get all questions in the group right.

We had a contract with Pearson education to develop Gradiance materials for my books and a number of other books. However, we eventually learned that they were unable to count on-line sales of our product, so the contract was cancelled. We have not been able to find sufficient sources of revenue to keep the company going as a profit-making enterprise, so we decided to make it free for all instructors who want to use it. We also allow students to register for the "omnibus courses" for each of the books. These courses let the students do all the homeworks we developed for the book. My home page http://i.stanford.edu/~ullman gives directions on how to get access to Gradiance.

I believe that in a few years, schools will be able to have students use well-designed courses for the entire CS curriculum, perhaps in many other disciplines as well.

Vision on knowledge dissemination

DJ: *You have made many of your books free for interested readers, Foundations of Computer Science, Mining of Massive Datasets. Why did you do this?*

JDU: Your readers should feel free to come to http://i.stanford.edu/~ullman/focs.html and http://i.stanford.edu/~ullman/mmds.html

Like our experience with Gradiance and automated homeworks, it is becoming harder to make money writing textbooks. File-sharing services have cut significantly into the market, as have on-line resellers of used texts. It appears that the idea of writing for royalties was an idea that had a brief run. 200 years ago, there was no such notion, and soon it will be a thing of the past. It seems academics are willing to put their intellectual property out their in the hope of developing a reputation rather than generating revenue. We've always done that with our research, so why not with our educational materials of all sorts. In fact, one of the most interesting developments along these lines is not the free book, but the free course. In the fall of 2011, three major Stanford courses – AI, Machine Learning, and Databases – are being offered to the public for free. The AI course has 140,000 students, and the other two over 70,000. Everything is automated, including a Gradiance reimplementation for managing homeworks and exams, and a discussion group where students answer the questions of other students, with no faculty involvement. Surprisingly, it all works. Well prepared students are willing to help those who are struggling.

I believe that in a few years, schools will be able to have students use well-designed courses for the entire CS curriculum, perhaps in many other disciplines as well. The role of faculty will change markedly. Instead of 1000 professors around the world, each preparing and teaching more or less the same thing at the same time, they will be able to devote their time to the things that really require personal involvement: helping students who are struggling, mentoring research students, doing their own research, and possibly creating educational materials of their own. And a consequence of this change is that we are finally going to be able to reduce the size of faculties. Every other industry has learned to do more with fewer people. It's now education's turn.

List of some classic books authored by Prof. Jeffrey D Ullman:

- Mining of Massive Data Sets (with A. Rajaraman), Cambridge Univ. Press, 2011 (available for free download through http://i.stanford.edu/~ullman/mmds.html)
- ***Database Systems:*** The Complete Book (with H. Garcia-Molina and J. Widom), Prentice-Hall, Englewood Cliffs, NJ, 2009.
- Introduction to Automata Theory, Languages, and Computation, (with J. E. Hopcroft and R. Motwani), Addison-Wesley, Reading MA, 1969, 1979, 2000.
- Elements of ML Programming, Prentice-Hall, Englewood Cliffs, NJ, 1993, 1998.
- A First Course in Database Systems (with J. Widom), Prentice-Hall, Englewood Cliffs, NJ, 1997, 2002, 2008.
- Foundations of Computer Science (with A. V. Aho), Computer Science Press, New York, 1992. C edition, 1994 (available for free download through http://i.stanford.edu/~ullman/focs.html)
- Principles of Database and Knowledge-Base Systems (two volumes), Computer Science Press, New York, 1988, 1989.
- ***Compilers:*** Principles, Techniques, and Tools (with A. V. Aho, M. Lam and R. Sethi), Addison-Wesley, Reading MA, 2007.
- Computational Aspects of VLSI, Computer Science Press, 1984
- Data Structures and Algorithms (with A. V. Aho and J. E. Hopcroft), Addison-Wesley, Reading MA, 1983.
- Principles of Compiler Design (with A. V. Aho), Addison-Wesley, Reading, MA, 1977.
- Fundamental Concepts of Programming Systems, Addison-Wesley, Reading MA, 1976.
- The Design and Analysis of Computer Algorithms (with A. V. Aho and J. E. Hopcroft), Addison-Wesley, Reading MA, 1974.

To be continued...