Jobs: Unix must grow or die

Joe Schallan, Contributing Editor

"I believe very strongly that Unix has got to become a mainstream operating system by 1990 or it's going to start to die," Apple Computer founder Steve Jobs told an audience of more than 1500 Unix programmers, developers, and users at the Summer Usenix Conference in Phoenix, Ariz., on June 10.

Jobs is now president and chairman of Next, Inc., which is developing a Unix-based workstation. He warned his listeners that, despite the size of its technical user community, Unix is not in the mainstream and although "some people believe we can go on sitting on the sidelines of what is considered to be the mainstream . . . . the world is changing and that's not going to be possible.

"I'm actually on your side now, since our whole new company is based on Unix," Jobs said, and he outlined what he thinks needs to be done in the next couple of years.

The Apple II operating system has an installed base of more than three million, MS-DOS more than six million, Macintosh more than one million, and OS/2 has a potential base of at least one million, Jobs noted. Unix, he said, has less than 250,000 systems installed.

Jobs said Unix developers should consider the players in the market before deciding how to make Unix a mainstream operating system. Neither IBM, Digital Equipment Corp., Apple, nor Microsoft are served by a successful Unix, he said.

Speaking of IBM, Jobs noted that, "when you're selling MIPS and gigabytes for as much as they are, it's not in your interest to have a lot of software out there that can run on machines from a lot of companies that are willing to sell MIPS and gigabytes at a lot less." DEC must protect a large investment in its proprietary operating system. Apple, with "the best user interface out there," is not "particularly interested in seeing a community of 1600 very bright people trying to copy that on top of Unix, especially if there are going to be multitasking Unixes all over." Microsoft has received large royalties from MS-DOS and looks forward to large OS/2 royalties. Microsoft's Xenix implementation of Unix is more of a "rearguard than a frontal attack," Jobs maintained.

The players on the other side—AT&T, Sun, Apollo, and universities—have less power in the marketplace, he said.

Comparing the two lists "should make most Unix enthusiasts a little nervous. . . . . There's a whole lot of money and a whole lot of energy riding on the side of not having Unix be successful."

But Unix can be successful, Jobs said, citing its advantages:

- It supports multitasking, virtual memory, and networking.
- Its development environment is "much more mature than many of its competitors."
- It is somewhat vendor-independent.
- It is probable that more programmers have been trained to use it than any other operating system.

However, the "bad news" about Unix, he said, are its shortcomings:

- There is more than one version, which "seems to cause everybody quite a bit of confusion."
- It is incomplete, lacking especially graphics and windowing.
- Its user interface "is impossible for mere mortals to use."
- It offers few end-user applications compared to PC operating systems.
- AT&T's $50-per-unit royalty makes Unix too expensive for low-end systems.

To solve these problems, Jobs said, the Unix architecture will have to evolve to include graphics and windowing and an applications toolkit.

Jobs said he believes the graphics will be Postscript-based. "Just as Postscript has become the standard in the printing industry, with IBM and DEC and Wang...

Correction

The report "Star Wars' Research Feeling Boycott?" (Soft News, March, pp. 94-95) said the Union of Concerned Scientists has about 100,000 member scientists and engineers. The actual number is about 20,000, a spokesman said. The group has about 80,000 financial supporters and nonscientist affiliates.

In this view of the Unix, OS/2, and Macintosh operating system environments, unimplemented levels are shaded. In his Usenix speech, Steve Jobs cited three points Unix enthusiasts should remember — that all the items in the Macintosh column are available today, that several billion dollars have been committed to unshading the OS/2 column, and that in the Unix column "we have two boxes to go."
Parallel, distributed processing lead NSF software research directions

The common theme of parallelism characterizes research across all the divisions of the Computer and Information Science and Engineering Directorate (CISE) at the National Science Foundation, said C. Gordon Bell, assistant director in charge of the directorate, in an interview with IEEE Software. Bell said he is acutely aware of the need for research in the software aspects of parallel processing.

Computer-science research programs under Bell have been criticized by some for ignoring software in favor of hardware and industrial programs. In this interview, Bell addresses the NSF's influence on software research and comments on research directions. The interview concentrates on parallel processing, an area critics said Bell deemphasized despite early indications he would stress it. The questions were posed in writing by Editor-in-Chief Ted Lewis, Contributing Editors Myers and Assistant Editor Galen Gruman.

Bell's directorate spent about $100 million on research in 1987 and hopes to spend $123 million in 1988, a 23-percent increase. Another $20 million is requested for administrative personnel and materials. (The NSF's funds went up about 20 percent in 1987 compared to 1986.) The NSF's research grants often set the tone for other, nondefense research, and the foundation's influence will be greater if Congress approves the 23-percent increase in NSF computing funds that President Reagan recently requested. The full 1988 budget request has survived the first of eight rounds in the congressional budgeting process, Bell said.

Bell's reputation was made at Digital Equipment Corp., where he was long-time vice president of engineering. He led the team that conceived the VAX architecture. More recently he was the chief technical officer at Encore Computer Corp.

Q: What areas of software research do you think will be the most vital in the next decade? Why?
A: Methods to design and build large programs and databases in a distributed environment are central. We have the opportunity and need for such programs through the availability of new powerful workstations, supercomputers, and mini-supercomputers. These are dramatically changing the way engineering and science is being carried out. We can now almost simulate most of the physical structures of interest to engineers and manufacturers ranging from manufacturing processes to molecular structures to VLSI chips.

Q: What software research areas is NSF funding now?
A: We fund what the [research] community considers to be important research, including object-oriented languages, databases, and human interfaces; semantics; formal methods of design and construction; connectionism; and data and knowledge bases, including concurrency. We aren't funding applications such as particular expert systems, unless they're potentially useful in another area of research being funded, such as VLSI design. Also, programming in the large is a concern — how do you write, evolve and share large programs?

Q: Do you see major shifts in software research directions taking place?
A: An article by Fred Brooks in the April 1987 issue of Computer presents various areas that are likely to contribute to improvement in software engineering. The gains look meager, so I don't expect dramatic shifts. I don't believe that software engineering is adequately taught in most places because the faculty haven't the experience nor do they appreciate the difficulties of management, training, and quality control in the process. Breakthroughs are hoped for and sought after.

I believe the big gains in software will come about by eliminating the old style of programming, by moving to a new paradigm, rather than magic tools or techniques to make the programming process better. Visicalc and Lotus 1-2-3 are good examples of a dramatic improvement in programming productivity. In essence, programming is eliminated and the work put in the hands of the users.

A similar opportunity exists for scientific and engineering computation in a program like MathCAD that, in essence, eliminates programming; it does not make programming in Fortran or C more productive or error-free.

These breakthroughs are unlikely to come from the software research community, because they aren't involved in real applications. Most likely they will come from people trained in another discipline who understand enough about software to be able to carry out the basic work that ultimately is turned over to the software engineers to maintain and evolve.

Q: How are distributed computing and artificial intelligence faring as research areas?
A: Both are of importance. AI is quite diffuse and should be segmented into its components. Many people argue that these areas are best pursued in terms of specific applications and objectives. A recent paper by [John] Hopcroft argued that robotics research is a major area for computer-science research. A research agenda, outlining the major problems and areas, would be useful for all of the computing community. Know anyone who would want to work on this?

Q: What areas appear to be poised to next capture the imagination and fervor of researchers?
A: Given the plethora of computers capable of generating vast arrays of numbers, research to use this performance to provide more insight is critical. In scientific computing we have an initiative in visualization — creative use of graphics — aimed at exploring these needs and opportunities. Also, accompanying the power is low-cost half-gigabyte CD PROMs and ROMs that should revolutionize the way we think about databases, books, handbooks, documentation, and computer-aided instruction as objects of computing research.

Some of the new machines are exciting and should be challenges in their own right because of the breakthroughs they provide. For example, the Connection Machine, which has 64K processing elements, carried out in about one hour all of the experiments in image processing that had been done in the last four decades.

Q: The recent Software Engineering Conference featured a strong division of opinion on mechanized programming. Some said that developing a programming system to write programs (called "process programming" at the conference) can automate much of the mundane tasks, while others warned it will lead students astray and damage the creative
part of programming. What do you think?
A: Mechanized programming is recreated and renamed every few years. In the beginning, it meant a compiler. The last time it was called automatic programming. A few years ago it was program generators and the programmer’s workbench. The better it gets, the more programming you do!

It isn’t unreasonable to believe that approaching software engineering from a purely mechanistic viewpoint can help, especially in managing the details of building large programs. Anything that helps and makes people more productive will be useful and will be assimilated. Arguments against change based on creativity are the same ones that were used to inhibit the use of high-level languages for building systems only a decade ago.

Q: What are good approaches to technology transfer?
A: Anything that works and gets the revolutions to take place. What I believe doesn’t work is having random congressmen decide that a certain machine should be built in their states and forcing an agency to buy a system when no real user would.

Q: Could you give us an example?
A: DoE [Energy Dept.] and DARPA [Defense Dept.’s Advanced Research Projects Agency] are two examples. [Bell declined to name the congressmen involved. —Ed.] The NSF has been able to operate under the peer-review system without such interference.

Q: Before we discuss parallel processing, is there anything of software research — in general and from NSF’s perspective — that we’ve missed?
A: You’ve covered just about everything except the opportunities and needs we have based on the mainline evolution of mini-supercomputers and supercomputers. Traditional software research has not played an important part in this, but it’s time, it’s not too late, to get involved.

[A. Nico] Habermann believes research in designing and documenting reusable software is one of the most fruitful areas of research to pursue vis-à-vis productivity and competitiveness — and I agree with him. Software is virtually the only engineering endeavor where one starts over each time a new artifact is to be built. I’m convinced that science and engineering computing itself is a good venue for doing first-class computing research.

Q: What is NSF’s role in software research in parallel processing?
A: We — together with our program advisory committees — have described the need for basic work in parallel processing to exploit both the research challenge and the plethora of parallel-processing machines that are available and emerging. We believe NSF’s role is to sponsor a wide range of software research about these machines.

This research includes basic computational models more suited to parallelism, new algorithms, standardized primitives (a small number) for addition to the standard programming languages, new languages based on parallel-computation primitives rather than extensions to sequential languages, and new applications that exploit parallelism.

Three approaches to parallelism are clearly here now: First, vector processing has become primitive in supercomputers and mini-supercomputers. In becoming so, it has created a revolution in scientific applications. Unfortunately, computer science and engineering departments are not part of the revolution in scientific computation that is occurring as a result of the availability of vectors. New texts and curricula are needed.

Second, message-passing models of computation can be used now on workstation clusters, on the various multicomputers such as the Hypercube and VAX clusters, and on the shared-memory multiprocessors (from supercomputers to multiple microprocessors). The Unix pipes mechanism may be acceptable as a programming model, but it has to be an awful lot faster for use in problems where medium-grain parallelism occurs. A remote procedure-call mechanism may be required for control.

Third, microtasking of a single process using shared-memory multiprocessors must also be used independently. On shared-memory multiprocessors, both mechanisms would be provided and used in forms appropriate to the algorithms and applications. Of course, other forms of parallelism will be used because it is relatively easy to build large, useful SIMD [single-instruction, multiple-data] machines.

Furthermore, it looks as if the programming will be quite straightforward because of the single thread of control. For example, a Connection Machine was just introduced with a 256M-byte memory, a 10G-byte disk operating at 40M bytes per second, direct-connected bitmapped memory for display, and the capability of calculating at 10 to 20 GFLOPS — or 10 times the speed of today’s largest supercomputer.

Alan Karp of IBM Research has offered a prize of $100 for a real scientific application if someone gets a speedup of 200 by 1995 using MIMD [multiple-instruction, multiple-data]. Measurement is key to achieving such a goal of parallelism. Unfortunately, the target is reached incrementally and not all at once.

To show you my own commitment to parallel processing, I would personally like to offer for the next 10 years, two $1000 annual awards for the best, operational scientific or engineering program with the most speedup (measured against a similar program run sequentially on one processor of the same system), not including vectorization on a vector processor.

The program must have a factor of two more speedup than a previous winning program. Operational is defined as a program used to produce a useful scientific or engineering result. The program should run at near the peak speed of any computer available (including various supercomputers), and be a cost-effective solution — no ‘‘toy’’ examples. One prize is for a program run on a general purpose computer system over $10 million, and the other is for any system. The rules should also comply with Karp’s. In fact, let me invite IEEE Software to flesh out the rules and run such a contest. [We’ve taken Bell up on his offer. The details are in the accompanying box. —Ed.]

Q: What performance do you expect from parallelism in the next decade?
A: Our goal is obtaining a factor of 100 in the performance of computing, not counting vectors, within the decade and a factor of 10 within five years. I think 10 will be easy because it is inherently there in most applications right now. The hardware will clearly be there if the software can support it or the users can use it.

Many researchers think this goal is aiming too low. They think it should be a factor of 1 million within 15 years. However, I am skeptical that anything more than our goal will be
too difficult in this time period. Still, a factor of 1 million may possibly be achieved through the SIMD approach.

The reasoning behind the NSF goals is that we have parallel machines now and on the near horizon that can actually achieve these levels of performance. Virtually all new computer systems support parallelism in some form (such as vector processing or clusters of computers). However, this quiet revolution demands a major update of computer science, from textbooks and curriculum to applications research.

Q: Critics complain that hardware and industrial research (high-speed communications, VLSI, industrial robotics, and parallel-processing machines) is dominating NSF parallel-processing spending. They say that the focus should be on software to use the hardware. How do you answer their concerns?
A: I generally agree with them, and that's why we are not, in future plans, emphasizing the design of new, parallel computers. Our new efforts will be mainly on two areas: first, using the hardware we have for research and, second, adding educational opportunities, including training students, especially undergraduates, in programming such machines.

Last year I took a survey at the biennial Snowbird Conference of computer-science and computer-engineering chairmen, and found that only 15 percent of the departments had environments for teaching parallel processing. To begin with, all departments should have such machines. I don't understand how people can do meaningful research without machines to test their theories or decent teaching without hands-on instruction.

Q: How much money is being spent on research on parallel-processing software?
A: In fiscal year 1986, about 25 percent of the 241 projects and 30 percent of the $16.6 million of support from the Computer and Computation Research Division were devoted to parallel-processing research. At most, 12 of the projects had hardware flavor. The CISE Institutional Infrastructure program (which used to be called CER [Coordinated Experimental Research]) funds 23 universities, seven of which are working primarily on parallel processing and nine of which have a secondary focus on parallel and distributed processing.

Q: How does this compare with hardware?
A: Only a small fraction of the funding is for new hardware research, including research into new architectures. None of our research borders on industrial or product research. It's quite basic.

However, NSF has a primary goal of improving US industrial competitiveness. This means training, experimental research with larger projects (as indicated in the president's recent statement initiating science and technology centers), and emphasizing areas like automated manufacturing.

The NSF Engineering Directorate also funds a significant amount of research in computing, especially the application of computers for robotics, control, engineering design, and neural computing. Its effort is more hardware-oriented than CISE's.

Q: To take full advantage of these new classes of parallel machines, what modifications of algorithms, languages, compilers, and programming environments may be needed?
A: The manufacturers provide primitives of all types to handle synchronization and communication. Most provide a Unix development environment for parallel processing. It would be great to get these primitives standardized across languages and machines so that texts could be written and undergraduate training could take off. Ada, for example, has the primitives for multitasking. The research community probably needs experience with what exists now before it starts to design a new language and environment.

Q: What applications do you see parallel processing being used for?
A: Scientific and engineering computation of all types can use all the processing power that can be developed for the foreseeable future. Many physical problems grow quadratically or cubically, and hence a factor of 1000 in processing is required to get an order-of-magnitude improvement in problem-solving.

In addition, many applications are inherently parallelizable, including transaction processing, message switching, commercial processing, human-interface management (like voice and video), database management, and robotics. High reliability depends on parallel processing.

Q: What do you think of claims from Unix creator Ken Thompson that parallel processing is impossible for people to create well, much less debug? ["Parallel Processing's Future Dim, Unix's Bright," Soft News, May, pp. 92-93.]
A: I believe the results obtained on the multiprocessors and multiprocessors believe this. People have to be trained to use the machines — it isn't that hard. Furthermore, training of this kind may encourage better decomposition for sequential programs. The greatest stumbling block in the way of learning parallel programming is the training people already have in thinking sequentially.

Q: How can parallel processing be harnessed for AI purposes?
A: It's unclear how much traditional AI applications are speeded up with parallelism. These may not be significantly different from conventional processing, so I expect a wide variation in the degree of useful parallelism. Some researchers believe that inherently parallel paradigms such as connectionism and neural modes of computing are necessary for revolutionary advances in most AI areas.
Software libraries: Real-world reuse

The cover articles this issue examine tools to make software reuse a reality. Many tools are new and have yet to be made part of day-to-day operations. When tools do exist, they are often scattered. However, despite the technology's infancy, some groups have built libraries for reusable components that are available. As examples of such libraries, these two reports detail an internal effort at Pacific Bell and a Defense Dept. cooperative effort.

Unix/C reusability library
Richard Anderson, Pacific Bell

A desire to eliminate redundant coding, reduce development and maintenance time, increase software reliability, and improve market potential prompted Pacific Bell to invest in a Unix/C reusability library for its employees. Since its implementation in 1986, the library has been expanded to help all members of the company's Unix/C community: system administrators, managers, development and maintenance staff, and users.

At the core of the library is an online archive of catalogued, reusable code and support information. Of particular interest to system and application developers — the library's original target users — are development and testing tools (like screen generators, database managers, and debuggers) and source-code modules ranging from special functions to full-scale applications. System-administration tools (like backup scripts, queue-management routines, and activity monitors) and project-management tools (like trouble-report systems and reminder services) appeal to other users. Text-processing tools, technical papers (product evaluations, standards, and system descriptions), training descriptions, and product descriptions have the widest appeal.

Most of the content has been contributed voluntarily by Pacific Bell employees; the rest comes from public-domain sources or from library staff members who generate items requested by Unix/C users. Everything in the library meets basic standards, such as adequate documentation within the source code and proper formatting. Once an item is made available, staff members will support it only if it has passed stringent quality-assurance tests.

All items have their contributors' identities and addresses so users can ask them the questions the library staff cannot answer.

Because most of the material is provided by Pacific Bell employees, the company has set up a system of incentives to encourage continued contributions. Tokens of appreciation (like coffee mugs and screwdrivers) are given to each contributor; heavy users and contributors of heavily used items are given larger gifts. These incentives supplement personal incentives, such as professional growth and recognition from colleagues, and the nature of the Unix environment, which promotes the creation and use of reusable modules.

Ada software repository
Richard Conn, General Electric

The Ada Software Repository on the Defense Dept.'s Defense Data Network contains more than 43 million bytes of Ada software and information, representing an investment of more than $5.5 million.

This software and information is available to US citizens and allies.
through the Defense Data Network itself, through computer networks connected via electronic mail gateways to the Defense Data Network (such as Bitnet, CSnet, and Usenet), and by magnetic tape.

Since the repository came on-line in November 1984, its files have been accessed more than 156,000 times by hundreds of military organizations, government research labs, corporations, and educational institutions. In late 1986, the Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office in the Pentagon adopted the repository as its interim software repository, pending the award of a future contract to establish a STARS Ada library.

The repository serves two roles: promoting the exchange and use of Ada programs (including reusable software-component libraries) and promoting the exchange of information on Ada-related topics. The more than 1000 software and documentation files in the repository are divided into 37 topics.

While the repository is supported by the Defense Dept., it is maintained and operated by Defense on an unfunded basis. The Defense Dept. contributes software from various projects, information, advice on legal issues, the host computer, and network interfaces. Volunteers from industry and academia contribute software, write technical reports, and maintain the repository.

Contents. Software in the repository ranges from reusable software components with less than 100 lines of code to application programs with more than 5000 lines. The reusable software falls under two categories: components and math. The components category contains parsers, dynamic string handlers, file I/O packages, garbage collectors, linked-list and queue manipulators, sort utilities, terminal emulators, and operating system interface routines. The math category contains routines for bit and set manipulation, logical operations on integers, matrix manipulation, and log, trig, exponential, and hyperbolic functions.

There are also topics covering the implementations of standards, including the Graphic Kernel System, the draft proposed ANSI/ISO database structured-query language standard, communications protocols (TCP/IP, SMTP, FTP, and Telnet), and software associated with the Common Ada Programming Support Environment Interface Set. Other topics include artificial intelligence, compiler benchmarks, software development tools (including cross-reference tools, a symbolic debugger, program-design-language tools, standards checkers, and "pretty printers"), project-management tools (including staff tracking, cost estimating, and project life-cycle modeling tools), and device-independent interfaces (such as forms generators and virtual terminals).

Limitations. The repository has some limits:

- Access to the Defense Data Network is required to acquire information and software easily and quickly.
- Access to the repository via electronic mail requires that certain software tools (file decompressor, database interpreter, and others) be established on the user's host computer.
- The software's quality is not consistent, so the user must establish a review mechanism to examine software imported from the repository before deciding whether to use it.
- The on-line documentation is the only way to determine what software is available.

If Unix users have accounts on the Defense Data Network, it may be weeks before their requests for information are fulfilled.

Continued from p. 101

and Apple [supporting it], . . . Postscript will become the standard for [Unix] screen-package graphics."

Unix applications must be able to be run in a vendor-independent environment. This is not now the case, Jobs pointed out, asserting that if it is not remedied by providing standardization up to the applications layer of Unix "we should take vendor independence off of our list of Unix positives."

Jobs summarized what Unix must look like to be a mainstream operating system: One standardized version that includes Postscript graphics and windows, and a user interface "for mere mortals," with no system-administration requirements. "You have to be able to take the Unix computer out of the box and plug it in, and be using it in 15 minutes," Jobs emphasized. And someone, he said, must build a "low-cost, breakthrough Unix computer."

Finally, Jobs said the Unix community must "evangelize third-party developers [to believe] that all of this is going to happen." Without action toward these goals, he cautioned, Unix could become "the Espenrito of the computer industry."