Systems Software Research is Irrelevant

Rob Pike
Bell Labs
Lucent Technologies
rob@plan9.bell-labs.com
Feb 21, 2000

A Polemic

This talk is a polemic that distills the pessimistic side of my feelings about systems research these days. I won't talk much about the optimistic side, since lots of others can do that for me; everyone's excited about the computer industry. I may therefore present a picture somewhat darker than reality.

However, I think the situation is genuinely bad and requires action.

Thesis

Systems software research has become a sideline to the excitement in the computing industry.

When did you last see an exciting non-commercial demo?

Ironically, at a time when computing is almost the definition of innovation, research in both software and hardware at universities and much of industry is becoming insular, ossified, and irrelevant.

There are many reasons, some avoidable, some endemic.

There may be ways to improve the situation, but they will require a community-wide effort.

Definitions

Systems

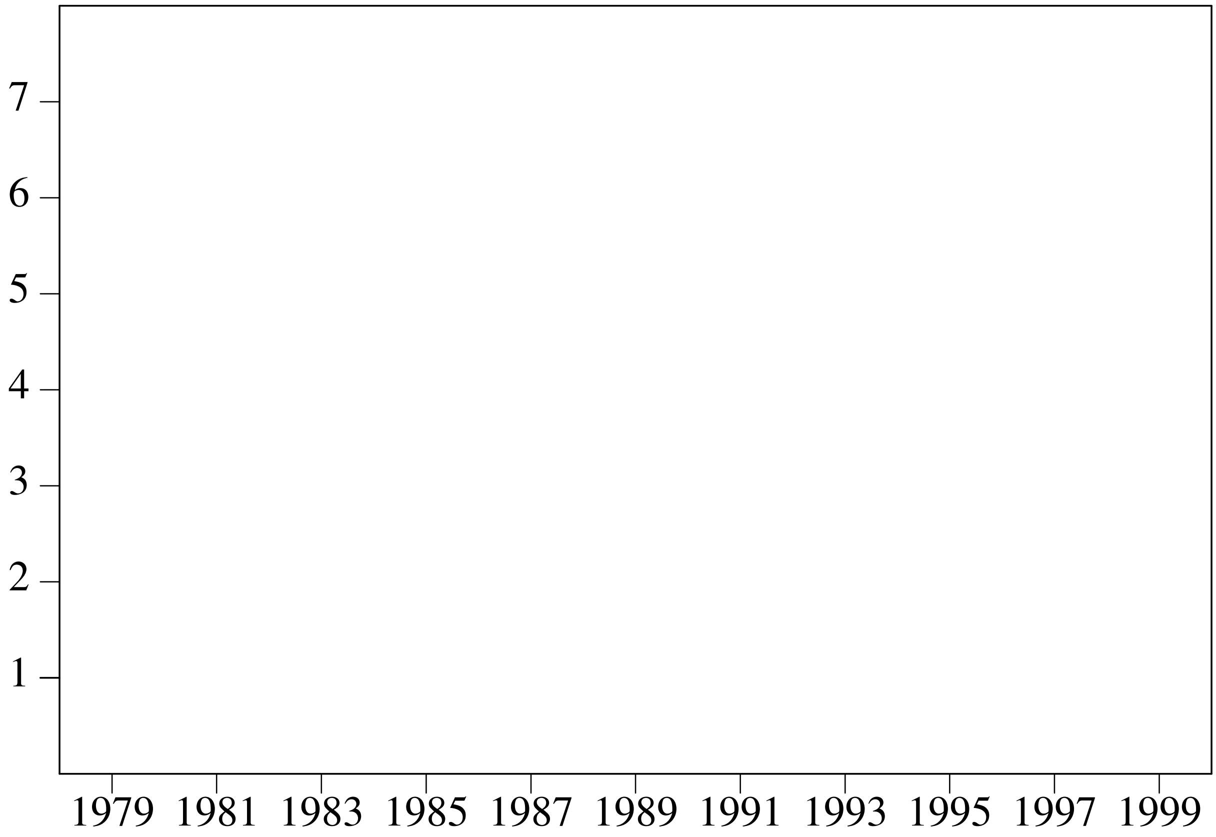
Software

Research

Is

Irrelevant

A Field in Decline



New Operating Systems at SOSP

"Who needs new operating systems, anyway?" you ask. Maybe no one, but then that supports my thesis.

"But now there are lots of papers in file systems, performance, security, web caching, etc.," you say. Yes, but is anyone outside the research field paying attention?

Systems Research's Contribution to the Boom

A high-end workstation:

1990	2000
Hardware	
33 MHz Mips R3000	600 MHz Alpha or Pentium III
32 megabytes of RAM	512 megabytes of RAM
10 Mbps Ethernet	100 Mbps Ethernet
Software	
Unix	Unix
X Windows	X Windows
Emacs	Emacs
TCP/IP	TCP/IP
	Netscape
Language	
C	C
C++	C++
	Java
	Perl (a little)

Hardware has changed dramatically; software is stagnant.

Where is the Innovation?

Microsoft, mostly. Exercise: Compare 1990 Microsoft software with 2000.

If you claim that's not innovation, but copying, I reply that Java is to C++ as Windows is to the Macintosh: an industrial response to an interesting but technically flawed piece of systems software.

If systems research was relevant, we'd see new operating systems and new languages making inroads into the industry, the way we did in the '70s and '80s.

Instead, we see a thriving software industry that largely ignores research, and a research community that writes papers rather than software.

Linux

Innovation? New? No, it’s just another copy of the same old stuff.

OLD stuff. Compare program development on Linux with Microsoft Visual Studio or one of the IBM Java/web toolkits.

Linux’s success may indeed be the single strongest argument for my thesis: The excitement generated by a clone of a decades-old operating system demonstrates the void that the systems software research community has failed to fill.

Besides, Linux’s cleverness is not in the software, but in the development model, hardly a triumph of academic CS (especially software engineering) by any measure.

What is Systems Research these days?

Web caches, web servers, file systems, network packet delays, all that stuff. Performance, peripherals, and applications, but not kernels or even user-level applications.

Mostly, though, it's just a lot of measurement; a misinterpretation and misapplication of the scientific method.

Too much phenomenology: invention has been replaced by observation. Today we see papers comparing interrupt latency on Linux vs. Windows. They may be interesting, they may even be relevant, but they aren't research.

In a misguided attempt to seem scientific, there's too much measurement: performance minutiae and bad charts.

By contrast, a new language or OS can make the machine *feel* different, give excitement, *novelty*. But today that's done by a cool web site or a higher CPU clock rate or some cute little device that should be a computer but isn't.

The *art* is gone.

But art is not science, and that's part of the point. Systems research cannot be just science; there must be engineering, design, and art.

What Happened?

A lot of things:

PC
Microsoft
Web
Standards
Orthodoxy
Change of scale
Unix
Linux
Startups
Grandma

PC

Hardware became cheap, and cheap hardware became good. Eventually, if it didn't run on a PC, it didn't matter because the average, mean, median, and mode computer was a PC.

Even into the 1980s, much systems work revolved around new architectures (RISC, iAPX/432, Lisp Machines). No more. A major source of interesting problems and, perhaps, interesting solutions is gone.

Much systems work also revolved around making stuff work *across* architectures: portability. But when hardware's all the same, it's a non-issue.

Plan 9 may be the most portable operating system in the world. We're about to do a new release, for the PC only. (For old time's sake, we'll include source for other architectures, but expect almost no one will use it.)

And that's just the PC as hardware; as software, it's the same sort of story.

Microsoft

Enough has been said about this topic. (Although people will continue to say lots more.)

Microsoft is an easy target, but it's a scapegoat, not the real source of difficulty.

Details to follow.

Web

The web happened in the early 1990s and it surprised the computer science community as much as the commercial one.

It then came to dominate much of the discussion, but not to much effect. Business controls it. (The web came from physicists and prospered in industry.)

Bruce Lindsay of IBM: HDLC $\subset$ HTTP/HTML; 3270s have been replaced by web browsers. (Compare with Visicalc and PC.)

Research has contributed little, despite a huge flow of papers on caches, proxies, server architectures, etc.

Standards

To be a viable computer system, one must honor a huge list of large, and often changing, standards: TCP/IP, HTTP, HTML, XML, CORBA, Unicode, POSIX, NFS, SMB, MIME, POP, IMAP, X, ...

A huge amount of work, but if you don't honor the standards you're marginalized.

Estimate that 90-95% of the work in Plan 9 was directly or indirectly to honor externally imposed standards.

At another level, instruction architectures, buses, etc. have the same influence.

With so much externally imposed structure, there's little slop left for novelty.

Plus, commercial companies that 'own' standards, e.g. Microsoft, Cisco, deliberately make standards hard to comply with, to frustrate competition. Academia is a casualty.

Orthodoxy

Today's graduating PhDs use Unix, X, Emacs, and Tex. That's their world. It's often the only computing world they've ever used for technical work.

Twenty years ago, a student would have been exposed to a wide variety of operating systems, all with good and bad points.

New employees in our lab now bring their world with them, or expect it to be there when they arrive. That's reasonable, but there was a time when joining a new lab was a chance to explore new ways of working.

Narrowness of experience leads to narrowness of imagination.

The situation with languages is a little better   many curricula include exposure to functional languages, etc.   but there is also a language orthodoxy: C++ and Java.

In science, we reserve our highest honors for those who prove we were wrong. But in computer science...

Change of scale

With so many external constraints, and so many things already done, much of the interesting work requires effort on a large scale. Many person-years are required to write a modern, realistic system. That is beyond the scope of most university departments.

Also, the time scale is long: from design to final version can be five years. Again, that's beyond the scope of most grad students.

This means that industry tends to do the big, defining projects operating systems, infrastructure, etc. and small research groups must find smaller things to work on.

Three trends result:

1. Don't build, measure. (Phenomenology, not new things.)

2. Don't go for breadth, go for depth. (Microspecialization, not *systems* work.)

3. Take an existing thing and tweak it.

I believe this is the main explanation of the SOSP curve.

Unix

New operating systems today tend to be just ways of reimplementing Unix. If they have a novel architecture and some do the first thing to build is the Unix emulation layer.

How can operating systems research be relevant when the resulting operating systems are all indistinguishable?

There was a claim in the late 1970s and early 1980s that Unix had killed operating systems research because no one would try anything else. At the time, I didn't believe it. Today, I grudgingly accept that the claim may be true (Microsoft notwithstanding).

A victim of its own success: portability led to ubiquity. That meant architecture didn't matter, so now there's only one.

Linux is the hot new thing... but it's just another Unix.

Linux   the Academic Microsoft Windows

The holy trinity: Linux, `gcc`, and Netscape.

Of course, it's just another orthodoxy.

These have become icons not because of what they are, but because of what they are *not*: Microsoft.

But technically, they're not that hot. And Microsoft has been working hard, and I claim that on many (not all) dimensions, their corresponding products are superior technically. And they continue to improve.

Linux may fall into the Macintosh trap: smug isolation leading to (near) obsolescence.

Besides, systems research is doing little to advance the trinity.

Startups

Startups are the dominant competition for academia for ideas, funds, personnel, and students. (Others are Microsoft, big corporations, legions of free hackers, and the IETF.)

In response, government-funded and especially corporate research is directed at very fast ‘return on investment’.

This distorts the priorities:

> Research is bent towards what can make big money (IPO) in a year.
>
> Horizon is too short for long-term work. (There go infrastructure and the problems of scale.)
>
> Funding sources (government, industry) perceive the same pressures, so there is a vicious circle.

The metric of merit is wrong.

Stanford now encourages students to go to startups because successful CEOs give money to the campus. The new president of Stanford is a successful computer entrepreneur.

Grandma

Grandma's on line. This means that the industry is designing systems and services for ordinary people.

The focus is on applications and devices, not on infrastructure and architecture, the domain of systems research.

The cause is largely marketing, the result a proliferation of incompatible devices. You can't make money on software, only hardware, so design a niche gimmick, not a Big New Idea.

Programmability   once the Big Idea in computing   has fallen by the wayside.

Again, systems research loses out.

Things to Do

Startups are too focused on short time scale and practical results to try new things. Big corporations are too focused on existing priorities to try new things. Startups suck energy from research. But gold rushes leave ghost towns; be prepared to move in.

Fiona's story: "Why do you use Plan 9?"

Go back to thinking about and *building systems.* Narrowness is irrelevant; breadth is relevant: it's the essence of *system*.

Work on how systems behave and work, not just how they compare. Concentrate on interfaces and architecture, not just engineering.

Be courageous. Try different things; experiment. *Try to give a cool demo.*

Funding bodies: fund more courageously, particularly long-term projects. Universities, in turn, should explore ways to let students contribute to long-term projects.

Measure success by ideas, not just papers and money. Make the industry *want* your work.

Things to Build

There are lots of valid, useful, interesting things to do. I offer a small sample as evidence. If the field is moribund, it's not from a lack of possibilities.

Only one GUI has ever been seriously tried, and its best ideas date from the 1970s. (In some ways, it's been getting worse; today the screen is covered with confusing little pictures.) Surely there are other possibilities. (Linux's interface isn't even as good as Windows!)

There has been much talk about component architectures but only one true success: Unix pipes. It should be possible to build interactive and distributed applications from piece parts.

The future is distributed computation, but the language community has done very little to address that possibility.

The Web has dominated how systems present and use information: the model is forced interaction; the user must go get it. Let's go back to having the data come to the user instead.

System administration remains a deeply difficult problem. Unglamorous, sure, but there's plenty of room to make a huge, even commercial, contribution.

Conclusions

The world has decided how it wants computers to be. The systems software research community influenced that decision somewhat, but very little, and now it is shut out of the discussion.

It has reached the point where I doubt that a brilliant systems project would even be funded, and if funded, wouldn't find the bodies to do the work. The odds of success were always low; now they're essentially zero.

The community universities, students, industry, funding bodies must change its priorities.

The community must accept and explore unorthodox ideas.

The community must separate research from market capitalization.