A Conversation with Wayne Rosing

Google is one of the biggest success stories of the recent Internet age, evolving in five years from just another search engine with a funny name into a household name that is synonymous with searching the Internet. It processes about 200 million search requests daily, serving as both a resource and a challenge to developers today.

Wayne Rosing brought his 30 years of experience in the computer industry to Google when he joined the company three years ago. Early in his career, Rosing

How the Web changes THE WAY

DEVELOPERS BUILD
AND RELEASE SOFTWARE.

held management positions at Data General and DEC, then was director of engineering for the Lisa and Apple II divisions

at Apple Computer. He spent nine years at Sun Microsystems, where he helped found Sun Labs, then joined FirstPerson, a wholly owned subsidiary of Sun, where he developed the technology base for Java. He then served as CTO and VP of engineering for Caere, manufacturer of

PHOTOGRAPHY BY TOM UPTON

optical character recognition (OCR) products. After Caere was sold, Rosing took time to pursue his lifelong interest in astronomy before coming back to the industry and joining Google.

He is questioned here by David J. Brown, senior staff engineer in the Solaris Engineering Group at Sun Microsystems. Earlier Brown was a member of the research staff at Stanford University, where he worked with Andy

of hardware. Eric Schmidt became the VP of software. It transmogrified fairly quickly into Eric and I sharing most of engineering. The one thing that I did in the first year that was really smart was getting the SPARC architecture established, getting SPARC machines going, and getting multiprocessor systems started with technology we imported from Xerox PARC, and then the SPARCstation products. I did that up until about 1991 or '92.

But I had been working more or less continuously since my early 20s, and I just wanted to take a break. My hobby is astronomy, so I went off and for a couple of years did a lot of astronomy. I established a research project in Chile that is still under way, surveying the southern galaxy and the interstellar medium in various color wavelength emission lines. I built a robotic telescope and generally had a lot of fun learning to do some professional scientific research. You know, it's a different discipline than engineering. I also built a number of special-purpose telescopes and various

Bechtolsheim on the prototype Sun workstation; was a founder of Silicon Graphics, where he developed early system and network software; and later established DEC's Workstation Systems Engineering Group in Palo Alto with Steve Bourne. During that time he completed a Ph.D. at Cambridge University. At Sun, Brown started and led the Application Binary Compatibility Program for Solaris. More recently he has worked on Sun's Linux-compatibility and open source strategies, and is now involved in the development of Sun's Web Services architecture.

David Brown First, your history. I understood that you were heading up Sun Labs from the conception. Then you went on to lead the FirstPerson effort—the Green Project, which later became Java. I'm also aware that you went off to work on an astronomy project. Maybe you can tell us a little bit about that and what brought you back out of semi-retirement into the active world again.

Wayne Rosing I joined Sun in 1985, essentially as VP

other engineering projects for a number of observatories around the country. I did that for a few years, and got that backlog of hobby stuff out of my system. Then I was getting sort of restless wanting to do the more traditional technical work that I was used to.

I went to a company called Caere, which is a manufacturer of OCR software. I was on Caere's board of directors, and I transitioned into being the VP of engineering for three years. That was a very fun time because, essentially, I really moved to managing software engineering and, more important, Windows-based consumer software. So that was a very interesting gearshift for me, to gain a different set of experiences.

Particularly when you're producing a CD—hundreds of thousands of them—and you put them out in distribution, you cannot have the kinds of errors that cause you to recall them, because that costs the company a lot of money. So it was an experience producing a big software release, once a year and on time, that had a lot of quality

assurance (QA) and that had very, very high quality.

That was a relatively different discipline from producing software that fundamentally you could update to the customer very rapidly.

And then Caere was purchased, and the company was moved east. I elected to stay here.

DB I was curious about how you came to Google.

WR It was done the classic way. A recruiter talked with Marian Cauwet, who was the director of engineering at FirstPerson, and then went on to be VP of engineering for Palm Computing. Marian gave him my name, on the guy's theory that I could give him names, because they were on a VP search.

Anyhow, he started telling me about the company, and I became very intrigued and said, "Hey, I would be very interested in talking to Google, because I think they're doing something really important."

And 13 weeks later, I was working here.

DB It's really interesting to hear you also talk about this evolution from the sort of software we were building in the past to these large-scale deployments or large-scale-release consumer-based software products, and the acute problems of software development surrounding the quality issues that we now face. That's certainly something we've been observing very intimately at Sun, surrounding Solaris, over the past 10 years.

And, really, it's changing all of our disciplines. In fact, when we were talking about this issue of *Queue*, the interest was to take a look at tools; one of the big things that we were thinking about was the problem of scale. Initially, we'd mostly been thinking about people wandering into large tracts of code, and how the heck they grapple with it.

But with Google, you've got this other huge area, which is the huge problem of scale for data. I thought I'd ask you a little bit about what you're doing in managing large amounts of storage and data and how that has impacted what goes on in your development at Google.

WR I think there are different dimensions. Sun built a platform that third parties build, arbitrarily, pieces of software upon—including banks and people who are really very, very serious about everything working right.

So Sun has to come out with a product, as represented with a software interface, for which, in some sense, the minimal acceptable standard is perfection. Now, we all know that you never really, truly achieve that. Because

you can't—Sun cannot possibly test every conceivable use that its software is put to—it's a pretty tough software engineering problem.

Caere represented a different dimension. Its product was an end-user product that parsed bits from a scanner and turned it into text. That's a very imprecise science, at best, but that software had to work reliably. And because we were a small company and we were using third-party distribution, we had economic constraints that basically said that the CD had to be perfect, in the sense that it would never experience a recall—again, an impossible task, but one that we got very close to achieving.

Google is very different. First of all, if we make a mistake, as soon as we see it on the site we can have the engineers go figure out the fix. We can push the software in a matter of hours, and we can update it. If we make a mistake on our own site, short of bringing the thing down—which, of course, we'll know instantly—we can fix things, because we don't have this problem of software recall or the associated revenue problems.

And, with the exception of our Google API—which is a rather small, experimental thing—people don't write applications on top of Google directly.

We are enjoying a relatively simple problem in producing software for the outside world. We have a few APIs. We will no doubt be adding more over time. But, at the moment, outside users at the software level aren't our significant problem.

DB One of the things that you're touching on is that the deployment method is really different. You have a kind of instantaneous deployment that hits everybody simultaneously, as compared with what we've done in the past.

WR Yes. And when I came here, I was 54 years old, so it took me a little bit of time to really grapple with what it meant to push software. I always thought of software as something you released, at the end of a year, with great pain and agony. At Google, we just push software all the time, so it's a very different notion. By the way, it's no less tolerant of sloppy engineering.

DB I imagine that the problems are really different. In fact, I think this has been one of the misconceptions about software, as compared with hardware: that we can continue to engineer it as it's running. I think in the past, where the deployment method was pushing CDs whose software was then installed on thousands of machines, we ran into some serious problems with scale, where you

find out just how hard it is to fix it once it's in the field.

What you're describing is the notion of it being in the field, but where there's a single point of contact—which is at your site. But you still have these problems of how to maintain stability in that system; obviously, your software engineering practices have to be acutely attentive to that problem.

WR There are lots of ways we get to the stability question. One thing to note is that a lot of engineering at Google is done on different timescales. Although we have had some projects that have taken multiple quarters, most projects reach “pushable,” or “prototypable,” or whatever deliverable state typically in a quarter or less. So I think an important part of how we do things is that there's a lot more incremental engineering. It's not the classic scenario where 500 people work for 18 months on a new release of Solaris, or the whole Linux community works for some extended period of time on the next major release of Linux.

Let's say we have a service that we want to enhance—we'll often just give the code to a team, and they will enhance it; they will write their version of the server. Then, after two or three rounds, the senior people will go back and generalize from the work done and will write a new abstraction that will pull the thing back together.

DB Maybe the way to look at it is that you're providing a bit more like an end-user application that's deployed over the Web. You touched on this distinction about deploying systems that offer quite general-purpose APIs.

And that certainly is very challenging. In fact, what you're doing is a lot more appliance-like versus building a general-purpose, programmatically offered engine.

WR As a matter of fact, our enterprise search solution, which is essentially Google.com in a yellow box, is an appliance. You plug it in, you give it an IP, it starts crawling your corporate intranet, and, voila, you're up and running with an enterprise search system. So it's very much an appliance in philosophy. We've actually learned to deploy Google.com in that mode.

It's primarily brought on by the fact that, in many cases, the cost of hardware is really irrelevant now. When you go to solve a certain class of software engineering

problems—particularly when you're talking PC-based commodity hardware—the game has changed.

By the way, there's quite a bit of additional, formal QA that is much more in the tradition of other companies that we put in place for the enterprise search appliance. There's a lot more discipline there.

Because the machine is crawling an intranet, we don't have any inherent ability to monitor it and look at it. The company may not want us to know all its technical

secrets. In that regard, we have to produce something that just works. We move more conservatively when it comes to our operating system. We tend to operate a revision or so back, relative to some of our most modern code, or the best stuff we're doing at the frontline at Google, because we really do need to build a robust product that essentially is in a box. In fact, that it's literally an appliance is a detail.

DB How does one go about testing these things, when you can't really see all of the applications to which they're put, or you can't see the context of their uses?

WR Well, our analogy would be to imagine a corporate intranet, into which you're putting a box that has to crawl a corporation that has evolved over 50 years with a computing infrastructure. That's a pretty tall order. Of course, we crawl the Web every day—and the Web is itself a completely dynamic, changing thing.

DB Well, perhaps that serves as a pretty good test case.

WR It's a pretty good test. But anyone who's in the search business knows the crawling team is always writing code. I mean, crawling is never going to be a solved problem; it's always going to be a work in progress because the Web changes in infinitely strange ways.

DB We talked a little bit about some of the things that are different about your system. Tell me about the key kinds of things you confront managing development, and what are the key challenges for your developers these days.

WR We have to write a lot of stuff ourselves. Just to give a few examples, we have a very large distributed file system

called GFS, for Google File System. We've had to do a fair amount of fundamental computer science in distributed systems, because we essentially have one of the largest distributed computers in the world.

I mean, we're talking about a lot of data. The unit of thinking around here is a terabyte. It doesn't even get interesting until there's more than many terabytes involved in problems. So that drives you into thinking of hundreds to thousands of computers as the generic way to solve problems.

The fundamental tools to do that kind of work aren't off the shelf. And you have to consider the spectrum of not only having to solve the problem, but also deploying it. And then we can't do it with an infinite number of people; we have to do it with a small number of people. We have to manage those machines when they're running in production, providing a service on a 24/7 basis.

DB I was chatting recently with Bob Sproull, director of Sun Labs' East Coast facility, and he was saying what's very interesting is the tooling that's going to make this stuff work. What are the things that you have to build up to make this stuff work automatically and without admin, so to speak?

WR The details are actually proprietary. But the basic notion is that when you have very large numbers of computers in multiple data centers, it's probably risky to attempt to manage this with human beings at the control panel. The management is done by software systems. We have some very good engineers who write those management systems.

When you get up to the level of routers and big co-location operations, then human beings get involved. But at the scale of our computing elements, most management now is automatic, all done programmatically. The hard part is writing those programs.

DB To what extent has commoditization helped out with that? You talked about hardware becoming basically free, and one of the things that one hears about Google is replacing x86 Linux boxes like lightbulbs when they fail. To what extent have you been able to exploit that dimension in what you do, to relieve the human element?

WR We've done that a lot. The fundamental principle is that stuff is going to break; therefore, engineer around it.

Now, if the stuff that breaks is cheap, then you can go with an N-plus-M redundancy model, and as long as the ratio of N and M is reasonable, you buy 100 machines

more, and then you don't have to send a technician out every day. You can send a technician out once a week or once a month. So, we monitor the failure rate; we try to minimize it, of course, but we engineer the systems to be tolerant of those types of failure rates. That allows a great deal of operational economy.

You have to remember, though, that search is a slightly different beast than running a bank. When you have to have absolute transactional integrity, the problem becomes a great deal more difficult. For instance, in the systems for billing our advertisers, parts of those systems are much more traditional and they're much more difficult to design because they have transactional integrity.

If you want to think of a huge distributed system, with a high degree of transactional integrity, there's a lot of really hard problems yet to be solved. We're just skirting around some of those problems now.

DB Can you tell us what some of those problems are?

WR Well, the most important one is how to build a large distributed database that has high transactional integrity—bank grade—and that's a very difficult problem.

An example where this comes to roost is in billing. Because of the nature of our business, which is showing ads that are part of a dynamic auction, we essentially have a micropayment billing system. There are millions of teensy little transactions per day. Then that all has to get reconciled, and brought into Oracle Financial Systems, and made to work, and be rock solid.

That's a very different problem than Sun. If Sun ships out a big enterprise server and forgets to invoice somebody, there's an interesting problem: A lot of money went out on the loading dock. So you don't have the huge numbers of transactions, and the transactions that you do have at Sun are large, as are the cost of goods sold. The cost of goods sold to show an ad is a small number. But you show millions of them.

So everything about this is very different. And databases don't quite know how to do this. You can't use an Oracle, for example, or a MySQL to roll this stuff up.

DB Because the granularity is so extremely different.

WR Right.

DB What you're saying is that you're just dealing with a very different kind of animal.

WR The point is that we write a lot of our own code. And

I would say we rewrite a lot of our own code. There's no such thing as an organized "Oh, let's do another Google." It's just that things get a little ponderous—a little too much spaghetti starts hanging out at the edges of something—and the engineers will say, "Let's fix it." Then a gang will form, and they'll go attack it.

So there's just this ongoing tension between writing new things and sort of refactoring the old work, and rewriting and evolving it.

It's remarkable in that the people who get interested in doing this come from all over engineering—sometimes newer people, sometimes old-timers, and sometimes mixtures of them. We don't have a rewrite group or a tools group here. We pretty much have a large concentration of very good engineers, and they tend to migrate toward the hard problems, wherever they may be. And those problems vary over time. So our engineering culture is actually very unstructured. It's far less structured than most companies I've seen.

DB Recently I've been thinking about the extent to which development and tools have shifted as a side effect of the evolution from more client/server oriented to this Web-deployed, multi-tier computing.

But I think what you're describing is a little different, because you're building more of a service that you're currently evolving. So perhaps what you're doing is constructing custom tools on the fly, in each generation, to solve the critical problems.

WR Right. And then extracting what we've learned into more generalized servers and libraries.

DB One of the points you brought up was about trying to architect these interfaces that are going to be programmatic and offered to third parties to build on top of. It's extremely hard to figure out what these interfaces should be until the system is deployed. So then you get into this whole conundrum: Now that we've figured it out through field experience, and we want to introduce a better-designed interface, how do you manage that legacy problem? That has been a frustration for us in doing traditional systems.

But it sounded like you were talking about being able to deploy a couple of different trial balloons and then abstracting from them as you go. I wonder if you could say a little bit more about how that works. Maybe it's because you're not exposing programming interfaces?

WR Well, that's right. They're exposed only to our own

people. And we don't have a hard rule that says you have to use what someone just did. You can take the code and modify it, run it on a different server.

Remember, we're just a start-up. We still have the urgency and excitement of a start-up here. We certainly are sober about our responsibility, and we're not sloppy. But there's a tremendous emphasis on, "Prototype it, get it out there, find out what our users think about it, and then start a cycle of improvement."

And so rapid iterative cycles, with lots of feedback from our users, is the game we like to play. Of course, that's really an engineer's dream.

But the trick is that you can't put out a prototype that doesn't work. It'll be all over the whole world. We have tremendous press visibility, so we have to do a good job. So, there's a lot of emphasis on testing inside—unit testing and other forms of regression testing.

DB And the dependencies start to bite you.

WR Well, it's getting harder, as our population gets larger, and there are more and more engineers working on the stuff, and more things being worked on.

At some point, the information can't get across the organization to the other side before it has become obsolete. There's some law about the exchange of information in large organizations, and I don't know what the diameter of the universe is for this—it's perhaps 1,000 engineers—and then we'll have to try something else.

DB Are there standard commercial tools that you guys like to use for your internal development? Or do you find that you're even building, for example, source code control, or time tools, or debugging tools?

WR Well, on the internal tool front, managing builds is always challenging. We have many engineers—they get frustrated, and they go fix things, and some of them are quite bright.

We use Perforce for source control, and that works well. As a general rule, we don't like to roll our own. That stuff is starting to get a little bit tricky now that we have three engineering sites, yet work on a common code base; the issues of how you do distributed development on a common code base are going to become more real. And, by the way, those are probably as much sociological as they are technological problems.

DB I would agree with that.

WR And when you've got a lot of sociology mixed in, it's not clear that any amount of technology can help you. You have to balance that stuff.

We use MySQL for a lot of things. And we've had to develop special techniques on how we use it, because of scale and distribution issues. Good old Linux, of course, just keeps on truckin' and does the job for us.

As a VP of engineering, I would normally be signing all kinds of big-tool stuff. It just doesn't happen. You know, the basic Linux core has been an extraordinarily great resource for Google.

We're just switching now to Red Hat Linux 9 as our main development environment, basically so that we get support. The Network File System (NFS) is another dimension to the environment. Generally, it works fine. We're starting to push the envelope there a little bit.

DB What do you see on the horizon, in terms of security, as it affects Google and what you're developing?

WR Well, the fashionable thing to say is that you don't put security in after you've engineered it.

You really, really need to start with a secure foundation. And in terms of developing—I'm not an expert in this area, but, for instance, we just recently deployed a Kerber-ized NFS. I guess that's a fairly new thing for Linux. So there are some very fundamental things that are happening in 2003, which strikes me as a little surprising, in terms of getting Kerberos, Lightweight Directory Access Protocol (LDAP), NFS, and Linux all sorted out. In this regard, you know better than I, Sun is way, way, way ahead of the Linux community in getting a lot of this stuff straight.

DB We kind of emerged into the enterprise commercial stage. And Linux is at an earlier stage in its life cycle than that.

WR Another area of concern in security is denial of service (DOS) attacks. This is something that we are very mindful of, because, obviously, our computing model is a honey pot for this kind of attacking.

We've done very well, so far. And that's another area where we've had to do some very sophisticated systems engineering.

DB So, people, as they're going forward and as they get more into the construction of services that are deployed over the Internet, really have to think about denial of service attacks and other things that could take down their systems. Are you finding that you're just attacking these problems on a fairly customized basis at the moment, or are there some themes that you can see in the ways that people can go about doing that?

WR There's a lot of work going on with some of the router companies and some of the network software and hardware companies. They recognize the need to be part of the solution. And then we do more on our own, so that we are able to manage things much more neatly. But that's an area where, again, I was a little surprised to find how primitive the state of the art is. I would just have intuitively expected the stuff to be very rock solid and well understood by now, and it isn't.

DB Until you can understand what the kinds of threats are that you're dealing with, it can be quite hard to know what the programming practices and disciplines are that you need to get engaged in to stop that.

WR Right.

DB You and I have been around the block here for a couple of decades; we've seen some major generational changes in the industry—the emergence of workstations and now the PC generation of computing and Web-type computing.

If you were to look back on this decade from 10 years on, could you say what your epitaph would be for the Google epoch?

WR Well, there is a driving theme that has excited me about computing since I was in high school, which was about 1962. I had read Vannevar Bush's paper on the *memex*. Although in 1962 the notion of a personal computer was, shall we say, foreign, the PDP-1 and PDP-5 [DEC's first mini-computers] existed, and it was obvious to me—and I wasn't even a technologist at the time—that the trend in computers was that they were going to get smaller. I knew enough physics to know—since light

travels at a constant speed—when they got smaller, they would get faster. I really, really care about small computers going quickly and getting them wired together.

I think the sum total of what I hope for the first decade of this century is some variant on the memex. We're going to have the vast majority of high-quality, permanent, high-value, human knowledge available to everyone, from many places, in multiple forms.

And that's fundamentally going to change humanity in as big a way as the printed word did—when it became inexpensive to replicate the printed word.

The other thing is computer translation. Inter-language translation is getting better—not as fast as anyone would like, but it's getting better, just as speech recognition is getting better. As we get more computer power, I think the important thing is that more human knowledge is going to exist in more languages.

DB One of the things that scares me most, looking forward, is the introduction of a huge amount of noise into the relatively small signal, and how we're going to come to grips with it.

WR I know what you mean. But you know what? Maybe the analogy is that people were concerned back in Gutenberg's time that we didn't really want the common people to know these things. Things were written in dead languages, so you had to be from the priesthood to know what was going on. It will always be disruptive to have massive amounts of new information. But, you know, people have always managed to figure out what is of value. Of course, it changes with the nature of economics and technology.

The day will come when Google won't be a search engine anymore, because everything will be searchable. So, instead, we'll have to algorithmically find you the good stuff. It will be an up-leveling of our ranking function, if you will, from what's the best document to what's the best, most well-formed knowledge on the subject.

I basically came to Google because it struck me what an incredibly neat thing this was to spend a large fraction of my remaining work years on.

I've spent 35 years learning how to do this stuff. And now I get to really put it to not only a challenging use, but an important one. ☺

© 2003 ACM 1542-7730/03/0900 \$5.00.