

## What's The Greatest Software Ever Written?

Witness the definitive, irrefutable, immutable ranking of the most brilliant software programs ever hacked.

By Charles Babcock, [InformationWeek](#)

Aug. 14, 2006

URL: <http://www.informationweek.com/story/showArticle.jhtml?articleID=191901844>

Most red-blooded technologists will offer a quick opinion on what's the greatest software ever, but when you take the time to evaluate what makes software truly brilliant, the choices aren't so obvious.

One of the most significant pieces of programming I know wasn't even software. Before the British built the Colossus machine, which translated German teletype code during World War II, it took the Allies up to six hours to decode a message and a day or more to pore over intelligence, draw conclusions, and pass along information to military command. After Colossus, the Allies gained a picture of German military activity across the English Channel as the day unfolded--intelligence that gave Gen. Dwight Eisenhower the confidence to launch the D-Day invasion.

Colossus was built in 1944 to perform Boolean operations on a paper data tape that streamed through the machine at 30 miles an hour. Its logic was literally wired into the machine. It is, perhaps, the greatest software that never got written.

So where does that leave us? First, let's set criteria for

what makes software great. Superior programming can be judged only within its historical context. It must represent a breakthrough, technical brilliance, something difficult that hadn't been done before. And it must be adopted in the real world. Colossus transformed a drawn-out mechanical process into electronics--it was an early computer--and provided a useful service by accelerating coded teletype translation. Colossus shaped history.

Another example of great programming was IBM's 360 system. The software was written as the first general-purpose computer operating system in 1964. Many of the truths we assume today about software--that simple designs are better than complicated ones, that a few skilled programmers will accomplish more than platoons of them--are captured in Frederick Brooks' book on the project, *The Mythical Man-Month* (Addison-Wesley Professional, 1995). Brooks already knew how many things could go wrong with big software projects before the 360 project began. In fact, he was a critic within IBM of carrying out the project at all; he thought it had too many potential points of failure. That's why IBM put him in charge of it, I suppose.

Wise that they did. The result was the first computer system capable of running different applications at the same time. It spawned the IBM line of mainframes, which evolved into the 370 Series and present zSeries. To this day, those systems remain backwardly compatible with Brooks' 360 operating system. Which leads me to another attribute of great software: It's got legs. It isn't easily superseded.

### We Know Great

Everybody agrees the IBM 360 was one of the greatest pieces of software ever written. Greatness is easier to assess given a long historical perspective. The closer you get to the present, however, the harder it is to name the greatest software.



**To the moon and  
back thanks to  
routine software**

Photo courtesy of NASA

Well, damn the torpedoes. With great insight, I've assembled this, my list of the greatest software ever written, from Colossus to the present. I've consulted software guru James Rumbaugh; Stuart Feldman, president of the Association of Computing Machinery; venture capitalists Ann Winblad and Gary Morgenthaler; Web site scripting software (PHP 3.0) authors Zeev Suraski and Andi Gutmans; and my little brother, Wally. The list remains my own, however. Those who find it full of wise and inspired choices can E-mail me at the address at the end of this story. For those who find it misguided, distasteful, or willfully ignorant, send your message to Wally, a 6-foot-3-inch former basketball star who still packs a wallop.

I've always been amazed at the Apollo spacecraft guidance system, built by the MIT Instrumentation Lab. In 1969, this software got Apollo 11 to the moon, detached the lunar module, landed it on the moon's surface, and brought three astronauts home. It had to function on the tiny amount of memory available in the onboard Raytheon computer--it carried 8 Kbytes, not enough for a printer driver these days. And there wouldn't be time to reboot in case of system failure when the craft made re-entry. It's just as well Windows wasn't available for the job.

The Apollo guidance system probably seems like routine software to technology sophisticates. Far more complex navigational systems are in operation today. The system's essentials were a few well-known algorithms based on proven logic. But to me, it's still rocket science. Great software dazzles us by virtue of what it does correctly in the face of everything that could go wrong.



**IBM 360 had staying power**

day.

To those unimpressed by the relative simplicity of the Apollo space system, I ask this: Would you rather place your life in the hands of a more complex system for moving things? Take, for example, the BAE Automated Systems software that was supposed to handle baggage at the Denver International Airport. On its October 1993 launch date, it lost or misdirected so much luggage, or delivered so much of it into the jaws of the conveyor, that the city had to delay opening the airport for 16 months. The cost overrun for the city: \$1.1 million per

For that matter, our lives already are in the hands of such software. The Federal Aviation Administration spent hundreds of millions of dollars, not once, but three times, trying to build an effective air traffic control system. It has thrown out about half of what it has created, technology valued at \$144 million, while the other half periodically hiccups and stalls. Great software? I'll stick with the Apollo guidance system. For software to be considered a success, it has to be up to handling the job it was created to do.

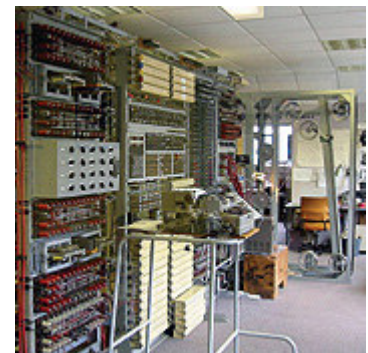
That axiom certainly applies to VisiCalc, the first spreadsheet software. It's great because it demonstrated the power of personal computing. The software put the ability to analyze and manipulate huge amounts of data into the hands of every business. But VisiCalc itself, despite representing a breakthrough concept, wasn't great software. It was flawed and clunky, and couldn't do many things users wanted it to do. The great implementation of the spreadsheet was not VisiCalc or even Lotus 1-2-3 but Microsoft Excel, which extended the spreadsheet's power and gave businesspeople a variety of calculating tools. Microsoft's claims that it makes great software are open to dispute, but the Excel spreadsheet is here to stay. Nearly everyone is touched by it.

### **Search for Intelligence**

There ought to be many examples of great software in the field of artificial intelligence. At one time, AI was going to produce a humanlike intelligence that could talk back to us, teach us things we didn't understand, and combine great reasoning power with command of endless data. What was the outcome for AI? Too much artifice, not enough intelligence.

Neural nets, which arose from AI research, produced automated fingerprint identification systems used worldwide. That's good pattern-matching, but is it actual intelligence? I don't see the logic.

The AI application that produced the first real breakthrough was the inference engine, a system with a knowledge base of conditions and rules. Such a computer can match a condition, such as a 104-degree fever in a patient, to a rule, such as the fact that bacterial infections cause high fevers. One of the best, the Mycin medical diagnosis system, could correctly identify bacterial infections in people based on their symptoms



65% of the time. That's better than most nonspecialized physicians. But it never moved out of the lab into popular use. No one knew who to sue when it was wrong.

**Colossus: The greatest software never written**

My favorite AI package was IBM's Deep Blue program, which defeated chess Grand Champion Garry Kasparov in a six-game match. Kasparov complained that Deep Blue had humans helping it behind the scenes, and he was right. IBM programmers were furiously revising Deep Blue between games to adjust it to Kasparov's playing style. That knocks Deep Blue off my list of candidates for great software. The IBM activity was within the rules but shouldn't have been. How was Kasparov supposed to compete? Rejigger his brain circuits?

AI software can be impressive, but all my examples fall short of being among the greatest.

I sometimes describe the browser as an emotionally handicapped dumb terminal. My brother, Wally, a research librarian, has convinced me that Mosaic, as the first graphical browser, "moved the Web out of the land of the tech weenies and into the world of ordinary humans." One predecessor, Gopher, is an example of a near miss, and then there was ViolaWWW, the first browser with buttons for moving backward and forward through Web pages.

But Mosaic's combination of address lines, mouse-based pointing and clicking, multimedia file displays, and hyperlinking in its window meant clients had finally found a perfect partnership with the information servers proliferating on the Internet. Mosaic combined elements for ease of use--the tool bar at the top and a set of pull-down menus--in a format that would be repeated in Netscape Navigator, Internet Explorer, and Firefox (in your Explorer window, select Help on the menu bar, click on About Internet Explorer, and a credit to Mosaic comes up). Technical brilliance? Not exactly, but a sorely needed, fresh technical synthesis. In other words, great software that opened the floodgates.

Can the same be said of the World Wide Web itself? Tim Berners-Lee produced a synthesis of hypertext linking, universal resource locator, and HTML page display that impacted our world, well, hugely. But the Web

copied existing ideas, which are all dependent on the underlying TCP/IP network protocols and BIND (Berkeley Internet Domain) domain name servers--the close-to-the-metal software that makes routers work. Nope, the Web isn't great software. But it sure pushes the needle off the scale when it comes to popular impact.

### **User's Delight**

Continuing into modern times, Google, in one aspect at least, represents great software. Search predated Google in the form of Lycos, Digital Equipment's AltaVista, and other engines. But Google incorporated a page-ranking structure into search results, assigning thousands of pages returned by the search engine a hierarchy reflecting their frequency of use. "The value of an academic paper is measured by the number of times it's mentioned in other papers and footnotes. Google adapted that convention to the Web," says Morgenthaler of Morgenthaler Ventures. It also moved a valuable information-structuring tool into the hands of millions of new users. That's great software.



### **Deep Blue had an artificial advantage--human tinkering**

Photo by Stan Hondal/AFP

I once thought of Sun's Java as a derivative language, a member of the C family that refined conventions already in existence. Upon reflection, I now know I was wrong. Java implemented the virtual machine on clients, allowing code to move over networks and run at a destination PC without knowing much about the machine itself. Java instituted the use of intermediate byte code, a form of source code that's been precompiled, which allows its translation to machine code just as it reaches the client. That equates to portability and performance. Java restricted the downloaded code to a sandbox, or set of boundaries--the client's hard drive, for example, was strictly off limits.

The sandbox saved users from security exposures they had experienced with unrestricted Microsoft Active X controls.

With these network-oriented features, Java swept into the business world at the dawn of the Internet era. Microsoft copied all of Java's best ideas when it made Visual Studio .Net. For Java to be first bitterly opposed, then embraced and extended, now that's a sure sign of greatness.

What about software that's a little more user-focused, like desktop publishing? Desktop publishing was made possible by Adobe Systems' PostScript, which could digitally format type and images either at a computer or inside a laser printer. Adobe had simplified the professional typesetting system that came out of Xerox's Silicon Valley research facility, Xerox Palo Alto Research Center, achieving the right blend of simplicity and clean operation in PostScript. It made desktop publishing commonplace. Nicely done, but not enough of a technical breakthrough to be called great software.

Speaking of Xerox Parc, the Apple Macintosh was based on the Alto system built there. Alto included the first windowing interface, the first mouse, and first unified graphical user interface. But it never made it to market. It took an Apple redesign to give it impact. I can still remember the first time I sat down at a Macintosh at a hole-in-the-wall computer shop in Endicott, N.Y. I got that "rocket science" feeling: I could see what it was doing, but I couldn't believe it. The Mac incorporated the power of object-oriented computing into the user interface, and users have never looked back. The first Mac operating system was great software.

### **As The Worm Turns**

Technology that infiltrates our daily lives and changes us qualifies as great. My next candidate meets that criterion, even if it's a loathsome piece of software. In 1988, the Morris worm raced around the Internet, infiltrating university servers and closing offices. Cornell student Robert Morris now says he made the worm so he could gauge the size of the Internet. Right.

Like most software, the worm theoretically could run in only one or two targeted environments, but it ended up illustrating something new about the Net. The worm could spread itself from server to server by exploiting a buffer overflow vulnerability in Sendmail. We didn't realize at the time how many back doors and defensive gaps lingered in Unix, Sendmail, Finger, and other systems. The worm also continuously queried servers and, in random cases, replicated itself. Morris said he added that feature to guarantee that his worm would spread. He succeeded.

As a piece of software, this intruder was a breakthrough, an eye-opening demonstration of what brilliant software might do on the anti-social side of the Internet. We were all starting to become interconnected; we all needed a wake-up call. Congratulations, Mr. Morris. The Great Worm proved an

incontrovertible alarm and accurate forecaster. It was great software.



**Was Sabre a winner because of science or bias--or both?**

Photo courtesy of American Airlines

American Airlines' Sabre system was great, showing how software could evolve beyond the tactical needs of business and into the strategic. Sabre had the ability to match a customer's travel needs with the flights available at a travel agent's office. Its listings also included flights from American's competitors. The system saved American and travel agents time and money, and it helped the carrier steal market share. American found that by giving its own flights higher position on search screens, they were selected more frequently, so it corrupted Sabre's search mechanisms to give its own flights priority. American called it "screen science." The U.S. government called it "screen bias" and banned the practice. Sabre blazed a path on both

counts: business strategy and business bias. With the advent of the Internet, searching for flights would reappear as Travelocity's customer self-service. And general-purpose search engines would implement pay-for-placement search results.

### **The Top Three**

So how do I rank my candidates on a list from 1-12? In descending order, the greatest software ever written is:

- 12. The Morris worm**
- 11. Google search rank**
- 10. Apollo guidance system**
- 9. Excel spreadsheet**
- 8. Macintosh OS**
- 7. Sabre system**
- 6. Mosaic browser**
- 5. Java language**
- 4. IBM System 360 OS**

That leaves my third, second, and top-most choices still to go. So here they are:

No. 3 is the gene-sequencing software at the Institute for Genomic Research. It isn't a mammoth software system, but "on sheer technical brilliance, it gets

10 out of 10," Morgenthaler says. The institute's sequencing system helped subdivide the task of understanding the DNA makeup of 20,000 human genes. Its breakthrough insights into the human genome and sequencing analysis, plus its ability to recombine subunits of analysis into the whole, "accelerated the science of genomics by at least a decade," Morgenthaler says. We now have the tools to begin tracing the paths of human migration out of Africa. The human genome reveals how minute the genetic differences are between ethnic groups at a time when such information is sorely needed. It gives a scientific basis for how humans can view each other as brothers at a time when we seem in danger of destroying one another. The software will be called on to perform many additional gene sequencing feats; the roots of many diseases and puzzles of heredity remain to be solved. Seldom have great research and great software been more closely intertwined.

My No. 2 choice is IBM's System R, a research project at the company's Almaden Research Lab in San Jose, Calif., that gave rise to the relational database. In the 1970s, Edgar Codd looked at the math of set theory and conceived a way to apply it to data storage and retrieval. Sets are related elements that together make up an abstract whole. The set of colors blue, white, and red, for example, are related elements that together make up the colors of the French flag. A relational database, using set theory, can keep elements related without storing them in a separate and clearly labeled bin. It also can find all the elements of a set on an impromptu basis while knowing only one unique identifier about the set.

System R and all that flowed from it--DB2, Oracle, Microsoft SQL Server, Sybase, PostgreSQL, MySQL, and others--will have an impact that we're still just beginning to feel. Relational databases can both store data sets about customers and search other sets of data to find how particular customers shop. The data is entered into the database as it's acquired; the database finds relationships hidden in the data. The relational database and its SQL access language let us do something the human mind has found almost impossible: locate a broad set of related data without remembering much about its content, where it's stored, or how it's related. All that's needed is one piece of information, a primary key that allows access to the set. I like System R for its incredible smoothness of operation, its scalability, and its overwhelming usefulness to those who deal with masses of data. It's software with a rare air of mathematical truth about it.

And now for The Greatest Software Ever Written--Unix.

Bell Labs often gets credit for creating the Unix operating system, but Bell never funded its development. In fact, the labs' management knew nothing about it. Bell Labs had committed developers to a multivendor project called Multics that made use of many new ideas for an operating system. But the project fell apart, and a Bell Labs participant, Ken Thompson, decided he wanted a personal version of Multics so he could write shoot-'em-up games, says Feldman (who was the No. 7 developer on the AT&T Unix project and is now president of the Association for Computing Machinery).

In the best tradition of software, Unix was an individual effort that took on a life of its own. Thompson crafted Unix on a Labs reject, a tiny DEC PDP 7 minicomputer with either 16 or 32 Kbytes of memory--Feldman isn't sure which. "Unix was written under great constraints," he says. "There was no memory and no CPU power. You'd be embarrassed today not to have more memory and CPU in your wristwatch."

Thompson designed his stripped-down operating system to move data in blocks or "pages" from a computer's random access memory onto disk, freeing up memory space. When they're needed again, the operating system knows to go to the disk and page them back into memory. This way, a big operating system can run on a small computer with a small amount of memory. His operating system also was a multiuser system. Even the mainframes of the day were limited to a single user, making computing time expensive. Thompson's Uniplexed Information and Computing System (Unics) would let two people use a computer at the same time.

The Computer Science Research Group at the Labs heard about Unics and wanted a copy. At the group's request, Thompson and a colleague, Dennis Ritchie, agreed they would add text formatting to their system, provided they were given a PDP 11/20, a larger machine. Thus, Unix text processing was born of barter. Unics became Unix; was recast in tighter, more portable C code; and was brought to market by AT&T as the Unix System III.

So Unix System III was the greatest piece of software--almost. Bear with me here.

## **A GNU Philosophy**

System III represented an advance, but it lacked many things, such as a windowing system, a graphical user interface, and a method for dealing with distributed systems. In an attempt to loosen IBM's stranglehold on large computers, AT&T made Unix available to researchers and universities for a small fee. Some people think open source code began when software became freely downloadable over the Internet. They're wrong. In fact, open source has its roots in the early distributions of Unix. One of those distributions came from a figure working into the night to improve the University of California at Berkeley's version. Other researchers would hear of Bill Joy's additions and ask him to send them a copy. So the first open source code wasn't a digital file. It was a reel of magnetic tape that Joy dropped in the mail late at night after finishing other work, according to Eric Allman, a grad student with Joy at Berkeley and the author of Sendmail. In 1977, the compilation of Joy's and other grad students' additions became known as the Berkeley Software Distribution, or BSD.

Unix was designed as discrete modules of code, each relating to a part of the hardware system. That made it easier to revise than IBM's operating systems. The Berkeley grad students made swift changes. They added a clean, fast file system, reliable networking, and the powerful vi code editor. They added Berkeley Sockets API, making it as easy to send data to a location on the network as to a local disk.

Defense contractor Bolt Beranek & Newman was at that time the official implementer of TCP/IP networking for the Defense Advanced Research Projects Agency. In BSD 4.1a, the Berkeley grad students modified TCP/IP to suit their own tastes. In 1986, Darpa would test the TCP/IP in BSD 4.3 and decide it was better than BBN's.

Bill Joy left Berkeley in 1982 to co-found Sun Microsystems, using the Berkeley Software Distribution as the basis for SunOS and Solaris. Sun and AT&T collaborated on improving System V, producing a consolidated System V Release 4. They agreed it would be the standard Unix for the future. AT&T, wanting a return on its Unix investment, increased its fees for the software.

But the Berkeley students weren't so easily derailed. They rewrote BSD Unix, ridding it of AT&T files and creating a new distribution that could run on low-cost Intel hardware. With yet another version looking like it might run off with AT&T's profits, AT&T's Unix Systems Labs sued BSDi, the company

distributing Berkeley Unix for Intel. Unix System Labs won an injunction against BSDi, tying up Intel-based Unix for years.

Meanwhile, high fees for Unix outraged Richard Stallman, a grad student who used it at the MIT artificial intelligence lab. Software, he decided, was an intellectual asset and should be free, like the published work of his fellow researchers. He set about building a set of tools called GNU that programmers could use to create their own software.

#### **MORE ONLINE ABOUT SOFTWARE BRILLIANCE**

What about windows? Find out at [informationweek.com/1101/windows.htm](http://informationweek.com/1101/windows.htm)

You tell us: Take our poll, nominate a candidate, and win (semi) valuable prizes at [informationweek.com/1101/software\\_poll.htm](http://informationweek.com/1101/software_poll.htm)

Software that almost made it at [informationweek.com/1101/list.htm](http://informationweek.com/1101/list.htm)

Those tools reached Linus Torvalds, a 21-year-old student in Helsinki, Finland, when he was looking for a version of Unix that would run on his Intel PC. He used them to develop the Linux kernel, and the rest is history. Linux became so popular that it displaced the Berkeley Software Distributions on Intel hardware. Today, Linux threatens to take over the high end of the market as well. But Linux is merely a clone of an incomplete GNU system and its BSD predecessors. They generated all the key concepts implemented in Linux. That's why Sendmail and BIND, building blocks of the Internet, were developed under Berkeley Unix, not System V. And that's why Microsoft, when it sought the best implementation of TCP/IP for Windows, took the one in BSD Unix. When Darpa wanted to build its Arpanet internetwork--or Internet--in 1983, it dropped its existing protocol and relied on BSD Unix TCP/IP.

So there you have it: The single Greatest Piece of Software Ever, with the broadest impact on the world, was BSD 4.3. Other Unixes were bigger commercial successes. But as the cumulative accomplishment of the BSD systems, 4.3 represented an unmatched peak of innovation. BSD 4.3 represents the single biggest theoretical undergird of the Internet. Moreover, the passion that surrounds Linux and open source code is a direct offshoot of the ideas that created BSD: a love for the power of computing and a belief that it should be a freely available extension of man's intellectual powers--a force that changes his place in the universe.



[www.wallstreetandtech.com](http://www.wallstreetandtech.com)

WallS  
Tec

Copyright © 2005 [CMP Media LLC](#)