List of software bugs

*Many software bugs are merely annoying or inconvenient but some can have extremely serious consequences – either financially or as a threat to human well-being. The following is a****list of software bugs****with significant consequences:*

**Space**

A booster went off course during launch, resulting in the destruction of NASA Mariner 1. This was the result of the failure of a transcriber to notice an overbar in a written specification for the guidance program, resulting in the coding of an incorrect formula in its FORTRAN software. (July 22, 1962).

The Russian Space Research Institute's Phobos 1 (Phobos program) deactivated its attitude thrusters and could no longer properly orient its solar arrays or communicate with Earth, eventually depleting its batteries. (September 10, 1988).

The European Space Agency's Ariane 5 Flight 501 was destroyed 40 seconds after takeoff (June 4, 1996). The US$1 billion prototype rocket self-destructed due to a bug in the on-board guidance software.

In 1997, the Mars Pathfinder mission was jeopardized by a bug in concurrent software shortly after the rover landed, which was found in preflight testing but given a low priority as it only occurred in certain unanticipated heavy-load conditions. The problem, which was identified and corrected from Earth, was due to computer resets caused by priority inversion.

In 2000, a Zenit 3SL launch failed due to faulty ground software not closing a valve in the rocket's second stage pneumatic system.

The European Space Agency's CryoSat-1 satellite was lost in a launch failure in 2005 due to a missing shutdown command in the flight control system of its Rokot carrier rocket.

NASA Mars Polar Lander was destroyed because its flight software mistook vibrations due to atmospheric turbulence for evidence that the vehicle had landed and shut off the engines 40 meters from the Martian surface (December 3, 1999).

Its sister spacecraft Mars Climate Orbiter was also destroyed, due to software on the ground generating commands in pound-force (lbf), while the orbiter expected newtons (N).

A mis-sent command from Earth caused the software of the NASA Mars Global Surveyor to incorrectly assume that a motor had failed, causing it to point one of its batteries at the sun. This caused the battery to overheat (November 2, 2006).

NASA's Spirit rover became unresponsive on January 21, 2004, a few weeks after landing on Mars. Engineers found that too many files had accumulated in the rover's flash memory. It was restored to working condition after deleting unnecessary files.

**Medical**

A bug in the code controlling the Therac-25 radiation therapy machine was directly responsible for at least five patient deaths in the 1980s when it administered excessive quantities of X-rays.

A Medtronic heart device was found vulnerable to remote attacks in March 2008.

**Tracking years**

The year 2000 problem spawned fears of worldwide economic collapse and an industry of consultants providing last-minute fixes.

A similar problem will occur in 2038 (the year 2038 problem), as many Unix-like systems calculate the time in seconds since 1 January 1970, and store this number as a 32-bit signed integer, for which the maximum possible value is 231 − 1 (2,147,483,647) seconds.

An error in the payment terminal code for Bank of Queensland rendered many devices inoperable for up to a week. The problem was determined to be an incorrect hexadecimal number conversion routine. When the device was to tick over to 2010, it skipped six years to 2016, causing terminals to decline customers' cards as expired.

**Electric power transmission**

The Northeast blackout of 2003 was triggered by a local outage that went undetected due to a race condition in General Electric Energy's XA/21 monitoring software.

**Administration**

The software of the A2LL system for handling unemployment and social services in Germany presented several errors with large-scale consequences, such as sending the payments to invalid account numbers in 2004.

**Telecommunications**

AT&T long distance network crash (January 15, 1990), in which the failure of one switching system would cause a message to be sent to nearby switching units to tell them that there was a problem. Unfortunately, the arrival of that message would cause those other systems to fail too – resulting in a cascading failure that rapidly spread across the entire AT&T long distance network.

In January 2009, Google's search engine erroneously notified users that every web site worldwide was potentially malicious, including its own.

**Military**

The software error of a MIM-104 Patriot, caused its system clock to drift by one third of a second over a period of one hundred hours – resulting in failure to locate and intercept an incoming missile. The Iraqi missile impacted in a military compound in Dhahran, Saudi Arabia (February 25, 1991), killing 28 Americans.

A Chinook crash on Mull of Kintyre in June 1994. A Royal Air Force Chinook helicopter crashed into the Mull of Kintyre, killing 29. This was initially dismissed as pilot error, but an investigation by Computer Weekly uncovered sufficient evidence to convince a House of Lords inquiry that it may have been caused by a software bug in the aircraft's engine control computer.

Smart ship USS Yorktown was left dead in the water in 1997 for nearly 3 hours after a divide by zero error.

In April 1992 the first F-22 Raptor crashed while landing at Edwards Air Force Base, California. The cause of the crash was found to be a flight control software error that failed to prevent a pilot-induced oscillation.

While attempting its first overseas deployment to the Kadena Air Base in Okinawa, Japan, on 11 February 2007, a group of six F-22 Raptors flying from Hickam AFB, Hawaii, experienced multiple computer crashes coincident with their crossing of the 180th meridian of longitude (the International Date Line). The computer failures included at least navigation (completely lost) and communication. The fighters were able to return to Hawaii by following their tankers, something that might have been problematic had the weather not been good. The error was fixed within 48 hours, allowing a delayed deployment.

**Media**

In the Sony BMG CD copy prevention scandal (October 2005), Sony BMG produced a Van Zant music CD that employed a copy protection scheme that covertly installed a rootkit on any Windows PC that was used to play it. Their intent was to hide the copy protection mechanism to make it harder to circumvent. Unfortunately, the rootkit inadvertently opened a security hole resulting in a wave of successful trojan horse attacks on the computers of those who had innocently played the CD. Sony's subsequent efforts to provide a utility to fix the problem actually exacerbated it.

**Video Gaming**

Eve Online's deployment of the Trinity patch, which erased the boot.ini  file from several thousand users' computers, rendering them unable to boot. This was due to the usage of a legacy system within the game that was also named boot.ini. As such, the deletion had targeted the wrong directory instead of the /eve directory.

The Corrupted Blood incident was a software bug in World of Warcraft that caused a status ailment, that was supposed to be locally restricted to a certain level of the game, to be set free, affecting all players everywhere in the virtual game world. This caused players to avoid crowded places in-game, just like in a "real world" epidemic, and the bug became the centre of some academic research on the spread of infectious diseases.

In the 256th level of Pac-Man, a bug results in a kill screen. The maximum number of fruit available is seven and when that number rolls over, it causes the entire right side of the screen to become a jumbled mess of symbols while the left side remains normal.

Valve's Steam client for Linux could accidentally delete all the user's files in every directory on the computer. This happened to users that had moved Steam's installation directory.  The bug is the result of unsafe shellscript programming:

STEAMROOT="$(cd "${0%/\*}" && echo $PWD)"

# Scary!

rm -rf "$STEAMROOT/"\*

The first line tries to find the script's containing directory. This could fail, for example if the directory was moved while the script was running, invalidating the "selfpath" variable $0. It would also fail if $0 contained no slash character, or contained a broken symlink, perhaps mistyped by the user. The way it would fail, as ensured by the && conditional, and not having set -ecause termination on failure, was to produce the empty string. This failure mode was not checked, only commented as "Scary!". Finally, in the deletion command, the slash character takes on a very different meaning from its role of path concatenation operator when the string before it is empty, as it then names the root directory.

**Encryption**

In order to fix a warning issued by Valgrind, a maintainer of Debian patched OpenSSL and broke the random number generator in the process. The patch was uploaded in September 2006 and made its way into the official release; it was not reported until April 2008. Every key generated with the broken version is compromised (as the "random" numbers were made easily predictable), as is all data encrypted with it, threatening many applications that rely on encryption such as S/MIME, Tor, SSL or TLS protected connections and SSH.

Heartbleed, an OpenSSL vulnerability introduced in 2012 and disclosed in April 2014, removed confidentiality from affected services, causing among other things the shut down of the Canada Revenue Agency's public access to the online filing portion of its website following the theft of social insurance numbers.

The Apple Computer, Inc. "goto fail" bug was a duplicated line of code which caused a public key certificate check to pass a test incorrectly.

**Transportation**

Toyota's electronic throttle control system (ETCS) had bugs that could cause sudden unintended acceleration.  At least 89 people were killed as a result.

The Boeing 787 Dreamliner experienced an integer overflow bug which could shut down all electrical generators if the aircraft was on for more than 248 days.

**Business**

The Vancouver Stock Exchange index had large errors due to repeated rounding. In January 1982 the index was initialized at 1000 and subsequently updated and truncated to three decimal places on each trade. This was done about 3000 times a day. The accumulated truncations led to an erroneous loss of around 25 points per month. Over the weekend of November 25–28, 1983, the error was corrected, raising the value of the index from its Friday closing figure of 524.811 to 1098.892.

**AT&T hangs up its long-distance service (1990):**For nine hours in January 1990 no AT&T customer could make a long-distance call. The problem was the software that controlled the company's long-distance relay switches—software that had just been updated. AT&T wound up losing $60 million in charges that day—a very expensive bug.

**The Pentium chip's math error (1993):**Thanks to a programming error, Intel's famous Pentium chip turned out to be pretty bad at math. The actual mistakes it made were fairly minute (beyond the eighth decimal point) and limited to certain kinds of division problems. But the irony—oh, the irony!—of a computer chip that made math errors made the problem blow up into the mother of all public relations disasters. After playing down the severity of the problem, causing even morepublic backlash, the company finally agreed to provide anyone who asked with a fixed chip.

**The Mars Climate Orbiter disintegrates in space (1998):**NASA's $655-million robotic space probe plowed into Mars's upper atmosphere at the wrong angle, burning up in the process. The problem? In the software that ran the ground computers the thrusters' output was calculated in the wrong units (pound–seconds instead of newton–seconds, as the NASA–Lockheed contract had specified). Fortunately software programs for subsequent missions to Mars have gotten the measurements right.

**Windows locks out nonsoftware pirates (2007):**For 19 hours on August 24, 2007, anyone who tried to install Windows was told, by Microsoft's own antipiracy software (called Windows Genuine Advantage) that they were installing illegal copies. If you'd bought Windows Vista, you discovered certain features shut off as punishment. The bug this time was both human and traditional: Someone accidentally installed a buggy, early version of the Genuine Advantage software on Microsoft's servers.

**Apple Maps gives us directions to nowhere (2012):**In its rivalry with Google, Apple decided to get rid of the much-adored Google Maps app that had always come on new iPhones—and to replace it with a new map app that Apple had written itself.

But in Apple's Maps whole lakes, train stations, bridges and tourist attractions were missing or mislabeled. The Washington Monument moved across the street. Riverside Hospital appeared in Jacksonville, Fla., even though it had become a Publix supermarket 11 years earlier. In the app's 3-D view bridges and dams seemed to melt into the water and Auckland, New Zealand's main train station was in the middle of the ocean.

The data that underlies mapping apps comes from dozens of different databases: for roads, satellite-photos, points-of-interest and so on. But meshing them together takes not just smart software, but also thousands of man-hours of handwork—which Google has had years to complete, but Apple hadn't. Little by little, Apple has been fixing these problems—but the company may find it difficult to instill all the lost trust in Maps.

**The Mars Climate Orbiter doesn't orbit**

Back in physics class, our teachers leaped all over answers that consisted of a number. If the answer was 2.5, they'd take their red pens and write "2.5 what? Weeks? Puppies? Demerits?" And proceed to mark the answer wrong.

Back then, we thought that they were just being pedantic. But it's the kind of error that can burn up a $327.6 million project in minutes. It did in 1998, when the Mars Climate Orbiter built by NASA's Jet Propulsion Laboratory approached the Red Planet at the wrong angle. At this point, it could easily have been renamed the Mars Climate Bright Light in the Upper Atmosphere, and shortly afterward been renamed the Mars Climate Debris Drifting Through the Sky.

There were several problems with this spacecraft -- its uneven payload made it torque during flight, and its project managers neglected some important details during several stages of the mission. But the biggest problem was that different parts of the engineering team were using different units of measurement. One group working on the thrusters measured in English units of pounds-force seconds; the others used metric Newton-seconds. And whoever checked the numbers didn't use the red pen like a pedantic high-school teacher.

The result: The thrusters were 4.45 times more powerful than they should have been. If this goof had been spotted earlier, it could have been compensated for, but it wasn't, and the result of that inattention is now lost in space, possibly in pieces.

**Mariner 1's five-minute flight**

On July 22, 1962, the first spacecraft of NASA's Mariner program blasted off on a mission to fly by Venus. The booster did its job, taking the spacecraft from its Cape Canaveral launchpad, but after a few minutes, Mariner 1 began to yaw off course. The guidance system failed to correct the trajectory, and guidance commands failed to correct it manually.

As the rocket veered off toward North Atlantic shipping lanes, the range safety officer did the only thing he could do: blow the thing up. Four minutes and 55 seconds into the mission, the Mariner 1 exploded.

NASA was already suffering from Sputnik envy, and the Mariner 1 incident was another international embarrassment for the agency. The postmortem of this debacle revealed what NASA described as "improper operation of the Atlas airborne beacon equipment" -- though later it came out that the mistranscription of a single punctuation mark by an engineer caused the mission's fatal software error.

In his 1968 book The Promise of Space, Arthur C. Clarke described the mission as "wrecked by the most expensive hyphen in history."

That may not be strictly accurate. Although NASA did mention a hyphen in some of its reports of the incident, it appears that the agency was simplifying the story for a nontechnical audience.

A more widely accepted account is that the punctuation mark was a superscript barover a radius symbol, handwritten in a notebook. In rocket science, the overbar signifies a smoothing function, so the formula should have calculated the smoothed value of the time derivative of a radius.

Without the smoothing function, even minor variations of speed would trigger the corrective boosters to kick in. The automobile driving equivalent would be to yank the steering wheel in the opposite direction of every obstacle in the driver's field of vision.

But few people know what an overbar is, and since it looks like a hyphen, that's how most people tell the story.

**Moth in the machine: Debugging the origins of 'bug'**

It's an oft-repeated tale that the grande dame of military computing, computer scientist and U.S. Navy Rear Admiral Grace Hopper, coined the terms bug and debug after an incident involving Harvard University's Mark II calculator.

The story goes like this:

On Sept. 9, 1945, a Harvard technical team looked at Panel F and found something unusual between points in Relay 70. It was a moth, which they promptly removed and taped in the log book. Grace Hopper added the caption, "First actual case of bug being found," and that's the first time anyone used the word bug to describe a computer glitch. Naturally, the term debugging followed.

Yes, it's an oft-repeated tale, but it's got more bugs in it than Relay 70 ever had.

For one thing, Harvard's Mark II came online in summer of 1947, two years after the date attributed to this story. For another thing, you don't use a line like "First actual case of bug being found" if the term bug isn't already in common use. The comment doesn't make sense in that context, except as an example of engineer humor. And although Hopper often talked about the moth in the relay, she did not make the discovery or the log entry.

The core facts of the story are true -- including the date of Sept. 9 and time of 15:45 hours -- but that's not how this meaning of the word bug entered the lexicon. Inventors and engineers had been talking about bugs for more than a century before the moth-in-the-relay incident. Even Thomas Edison used the word. Here's an excerpt from a letter he wrote in 1878 to Theodore Puskas, as cited in [The Yale Book of Quotations](http://yalepress.yale.edu/yupbooks/qyd/) (2006):

'Bugs' -- as such little faults and difficulties are called -- show themselves and months of intense watching, study and labor are requisite before commercial success or failure is certainly reached.

Word nerds trace the word bug to an old term for a monster -- it's a word that has survived in obscure terms like bugaboo and bugbear and in a mangled form in the word boogeyman. Like gremlins in machinery, system bugs are malicious. Anyone who spends time trying to get all the faults out of a system knows how it feels: After a few hours of debugging, any problems that remain are hellspawn, mocking attempts to get rid of them with a devilish glee.

And that's the real origin of the term bug. But the tale of the moth in the relay is worth retelling anyway.

**Forty seconds of Ariane-5**

The European Space Agency (ESA) has also suffered embarrassment on the software front. The inaugural flight of its fifth-generation Ariane launcher bested NASA's Mariner 1 score for unmanned spacecraft disaster: It took only 40 seconds to blow up.

On June 4, 1996, after the kind of dramatic vertical blastoff you'd expect from a high-profile European vehicle, cameras on the ground barely had time to focus on the Ariane-5 as it turned around and began to fall apart, before it completely exploded.

The Ariane Flight 501 disaster began with a loss of guidance and attitude information 30 seconds after liftoff. Once it veered completely off course, it automatically self-destructed.

The problem was that Ariane-5's inertial reference system dealt with 64-bit floating-point data and converted it into 16-bit signed integer values. The result of the data conversion was too large for a 16-bit signed integer, which caused an arithmetic overflow in the hardware. In the ESA's case, a software handler that could have dealt with the problem had been disabled, and so there was no levee to dam the cascade of system failures that led to the destruction.

**Pentium chips fail math**

In 1994, an entire line of CPUs by market leader Intel simply couldn't do their math. The Pentium floating-point flaw ensured that no matter what software you used, your results stood a chance of being inaccurate past the eighth decimal point. The problem lay in a faulty math coprocessor, also known as a floating-point unit. The result was a small possibility of tiny errors in hardcore calculations, but it was a costly PR debacle for Intel.

How did the first generation of Pentiums go wrong? Intel's laudable idea was to triple the execution speed of floating-point calculations by ditching the previous-generation 486 processor's clunky shift-and-subtract algorithm and substituting a lookup-table approach in the Pentium. So far, so smart. The lookup table consisted of 1,066 table entries, downloaded into the programmable logic array of the chip. But only 1,061 entries made it onto the first-generation Pentiums; five got lost on the way.

When the floating-point unit accessed any of the empty cells, it would get a zero response instead of the real answer. A zero response from one cell didn't actually return an answer of zero: A few obscure calculations returned slight errors typically around the tenth decimal digit, so the error passed by quality control and into production.

What did that mean for the lay user? Not much. With this kind of bug, there's a 1-in-360 billion chance that miscalculations could reach as high as the fourth decimal place. More likely, with odds of 1-to-9 billion against, was that any errors would happen in the 9th or 10th decimal digit.

But wouldn't you know it? A Virginia-based math professor named Thomas Nicely needed that level of accuracy, found he wasn't getting it and figured out why.

In October 1994, he alerted Intel, then others, to the problem. Intel retorted with a response only marginally less tactful than "Oh, that thing? Yeah, we noticed that back in June."

Thus began an inexorable slide into PR hell and a costly mop-up bill. In January 1995, Intel announced a pretax charge of $475 million against earnings, most of which apparently stemmed from replacing flawed processors.

The bottom line in this arithmetic mess is this: In lookup-table and money calculations, 1,066 – 5 = –$475,000,000. Any way you look at it, that's bad math.

**More math bugs**

Intel's Pentium flaw wasn't the only math-related bug to cause a PR disaster. These two had Microsoft execs red in the face:

Windows Calculator 3.x: In 1994, a bug in CALC.EXE came to light that had quietly been kicking around since Windows 3.x first appeared in 1990. Propellerheads had fun subtracting 2.1 from 2.11 in Windows Calculator and getting an answer of not 0.01, but 0.00.

Excel 2007: Ask people with calculators or slide rules to multiply 850 x 77.1, and they'll answer 65,535. But in September 2007, it was discovered that Excel 2007 answered 100,000. According to Microsoft, this bizarre rounding-up occurred only in calculations that resulted in 65,535 or 65,536. What's more, Excel actually calculated the correct answer, but a bug prevented it from displaying properly.

**Call waiting ... and waiting ... and waiting**

On Jan. 15, 1990, around 60,000 AT&T long-distance customers tried to place long-distance calls as usual -- and got nothing. Behind the scenes, the company's 4ESS long-distance switches, all 114 of them, kept rebooting in sequence. AT&T assumed it was being hacked, and for nine hours, the company and law enforcement tried to work out what was happening. In the end, AT&T uncovered the culprit: an obscure fault in its new software.

Here's how the switches were supposed to work: If one switch gets congested, it sends a "do not disturb" message to the next switch, which picks up its traffic. The second switch resets itself to keep from disturbing the first switch. Switch 2 checks back on Switch 1, and if it detects activity, it does another reset to reflect that Switch 1 is back online. So far, so simple.

The month before the crash, AT&T tweaked the code to speed up the process. The trouble was, things were too fast. The first server to overload sent two messages, one of which hit the second server just as it was resetting. The second server assumed that there was a fault in its CCS7 internal logic and reset itself. It put up its own "do not disturb" sign and passed the problem on to a third switch.

The third switch also got overwhelmed and reset itself, and so the problem cascaded through the whole system. All 114 switches in the system kept resetting themselves, until engineers reduced the message load on the whole system and the wave of resets finally broke.

In the meantime, AT&T lost an estimated $60 million in long-distance charges from calls that didn't go through. The company took a further financial hit a few weeks later when it knocked a third off its regular long-distance rates on Valentine's Day to make amends with customers.

**Windows Genuine Disadvantage**

Introduced in 2006, Windows Genuine Advantage was never a popular initiative with Microsoft's customers. Consumers had trouble seeing the advantages: It did nothing to help the security or stability of a legitimate Windows installation. All it did was help Microsoft root out software piracy.

In that task, it was as vigilant as, well, a vigilante. In fact, in late-August 2007, it found piracy everywhere it looked -- even among thousands of legitimate Windows customers.

On Friday, Aug. 24, someone on the WGA team accidentally installed bug-filled preproduction software on the WGA servers. The team quickly rolled back to a tested release of the software, but they didn't check that their fix actually addressed the problem. It didn't. So for 19 hours, until around 3 p.m. the following day, the server flagged thousands of WGA clients across the globe as illegal.

Windows XP customers were told they were running pirated software. Windows Vista customers were slapped harder: They had features turned off, including the eye candy Aero theme and support for ReadyBoost virtual RAM drives.

The first official response to complaints didn't help much: Disgruntled patrons were advised to try to revalidate on Tuesday. But even when the problem was fixed, mid-Saturday afternoon, Vista clients still had to revalidate their Windows installations before they could ReadyBoost their way back into Aero.

OK, so this was a relatively mild issue in engineering terms, and strictly speaking, it was caused by human error. But the error in question was deploying buggy, untested software, and when you factor in the number of people affected, the level of anger induced and the knock-on effect of bad publicity, it was more severe than it seems at first glance.

**Patriot missile mistiming**

During the first Persian Gulf war, Iraqi-fired Scud missiles were the most threatening airborne enemies to U.S. troops. Once one of these speeding death rockets launched, the U.S.'s best defense was to intercept it with an antiballistic Patriot missile. The Patriot worked a bit like a shotgun, getting within range of an oncoming missile before blasting out a cloud of 1,000 pellets to detonate its warhead.

A Patriot needed to deploy its pellets between 5 and 10 meters from an oncoming missile for the best results. This requires split-second timing, which is always tricky with two objects moving very fast toward each other. Even the Patriot's most prominent booster, then-President George H.W. Bush, conceded that one Scud (out of 42 fired) got past the Patriot. The single failure the president acknowledged was at a U.S. base in Dhahran, Saudi Arabia, on Feb. 25, 1991, and it cost 28 soldiers their lives. The fault was traced to a software error.

The Patriot's trajectory calculations revolved around the timing of radar pulses, and they had to be modified to deal with the high speed of modern missiles. A subroutine was introduced to convert clock time more accurately into floating-point figures for calculation. It was a neat kludge, but the programmers did not put the call to the subroutine everywhere it was needed. High-speed trajectories based on one accurately timed radar pulse and one less-precise time increased the chances of poorly timed deployment.

Apparently, the issue was known, and a temporary fix was in place: Reboot the system every so often to reset the clocks. Unfortunately, the term "every so often" wasn't defined, and that was the problem in late February at Dhahran. The system had been running for 100 hours, and the clocks were off by about a third of a second. A Scud travels half a kilometer in that time, so there was no chance the Patriot could have intercepted it.

On a side note, some experts did dispute the president's claimsof a more than 97% success rate for Patriots vs. Scuds, so it's possible that this bug caused more (but less high-profile) damage than the incident at Dhahran.

**Therac-25 Medical Accelerator disaster**

Radiation therapy is a handy tool in the fight against some contained forms of cancer: Beams of electrons zap the bad stuff, and the body disposes of the dead matter. It has a strong success rate, but it depends on accurate aim and focus. That's something that the medical world leaves to machinery. Unfortunately for six patients between 1985 and 1986, the Therac-25 was the machine in question.

The Therac-25 handled two types of therapy: a low-powered direct electron beam and a megavolt X-ray mode, which required shielding and filters and an ion chamber to keep the dangerous beams safely on target. The trouble was that the software that powered the unit was repurposed from the previous model, and it wasn't adequately tested.

If the operators changed the mode of the device too quickly, a race condition occurred: Two sets of instructions were sent, and the first one to arrive set the mode. In six documented cases, this meant that megavolt X-rays were sent, unfiltered and unshielded, toward patients requiring direct electron therapy. At least two of them screamed in pain and tried to run from the room. All of them suffered radiation poisoning, which claimed several lives.

The Therac-25, which was recalled in 1987, has become an object lesson in what can go wrong with powerful medical machinery. The code didn't cause overdoses in earlier Therac models because hardware constraints prevented them. Reusing code on a new system without thorough testing is a programming no-no, with good reason.

The new system did deliver error messages during race-condition events, but the codes were cryptic, undocumented and easily overridden -- which is what operators did. With adequate documentation and training, the overdoses would never have happened. Additionally, a smaller bug that set up flag variables occasionally caused arithmetic overflows that bypassed safety checks.

**Multidata Systems/Cobalt-60 overdoses**

Unfortunately, the Therac-25 disaster wasn't the last software-related radiation therapy failure. Fifteen years after the Therac-25 incident, a Cobalt-60 machine in Panama's National Cancer Institute overdosed more than two-dozen patients with gamma radiation.

As with the Therac-25, the Cobalt-60 system was an accident waiting to happen. Unlike the Therac-25, the Cobalt-60 was an old, overused and undermaintained piece of hardware. The software that ran it was an aftermarket program from Multidata Systems, because the Panamanian hospital could not afford what the machine's manufacturer, Theratronics, charged.

Two of the technicians who operated the Cobalt-60 had quit, leaving the rest to work 16-hour days to keep up with treatments. Very sick patients would sometimes wait four to six hours a day for scheduled treatments.

Overworked and tired technicians requested some software maintenance, but management overlooked their requests. Somewhere along the line, the technicians hit upon a more efficient way to line up the shields that defined the radiation's target. It wasn't in the manual, but it seemed to work. Unfortunately, if you lined up the shields in a particular order, an obscure bug in the Multidata software meant that the patients were overirradiated. Because of massive overwork and undersupervision, the process went on for seven months.

By the time Multidata Systems issued an advisory about a "data entry sequence that creates a self-intersecting shape outline" in mid-2001, it was too late for many patients. The exact death toll is hard to calculate -- these were very sick patients even before their treatment -- but it's a tragic mess-up by any measure.

**Osprey aircraft crash**

Two weeks before Christmas in 2000, a U.S. Marine Corps Osprey, a hybrid airplane and helicopter, suffered a hydraulic system fault that should have been remedied without loss of life. A hydraulic line broke in one of the two engine cases as the Osprey was shifting from airplane to helicopter mode for landing.

According to the Marine Corps major general who presented reports during the investigation of the incident, the trouble was "compounded by a computer software anomaly." The flight-control computer stopped the rotation of the engine pods when it detected the hydraulic failure.

The pilots went through the normal procedure and pressed the primary reset button to re-engage the pods. At this point, both prop rotors went through "significant pitch and thrust changes," which led to a stall. The plane crashed into a marsh and killed all four Marines onboard.

The nature of the software flaw is still hard to track down: Boeing and Bell Helicopter made the Osprey, and Boeing's spokesman said only that changes were made in the software. Requests for details were referred to the government, and as of now, the explanation has not been forthcoming.

**End-of-the-world bugs**

Remember how the world descended into nuclear oblivion on Sept. 23, 1983? No? Well, thank your lucky stars -- this is a tale of bugs so major they could have brought the entire world to a standstill.

It was all averted by the common sense of one individual, who ignored the Soviet early-warning system's faulty reports of incoming missiles and didn't launch a counterattack on the United States.

The warning system set off klaxons at half past midnight on that September morning. Apparently, the U.S. had launched five nuclear missiles toward what the U.S. president had taken to calling "the Evil Empire."

At the time, Lt. Col. Stanislaus Petrov reasoned his way to a decision not to respond: The USSR was in a shouting match with the U.S. about a Soviet attack on Korean Air Lines Flight 007 three weeks earlier, but it was only a rhetorical battle at that stage. Besides, if the U.S. wanted to attack the Soviet Union, would it really launch only five missiles?

Petrov ordered his men to stand down, and 15 minutes later, radar outposts confirmed that there were no incoming missiles. The decision took less than five minutes, it was confirmed within half an hour, and the world remained at peace.

When the early-warning system was later analyzed, it was found to have more bugs than a suburban compost heap -- which meant that although Stanislaus Petrov had saved the world, he'd made a serious error of judgment: He had shown up the incompetence of Soviet programmers.

This was not good for morale, or for the lieutenant colonel. He was cold-shouldered into an early retirement and was largely unsung until May 21, 2004, when a San Francisco-based organization called the Association of World Citizens bestowed its highest honor -- world citizenship -- and a financial reward on him.

July 28, 1962 -- Mariner I space probe. A bug in the flight software for the Mariner 1 causes the rocket to divert from its intended path on launch. Mission control destroys the rocket over the Atlantic Ocean. The investigation into the accident discovers that a formula written on paper in pencil was improperly transcribed into computer code, causing the computer to miscalculate the rocket's trajectory.

1982 -- Soviet gas pipeline. Operatives working for the Central Intelligence Agency allegedly plant a bug in a Canadian computer system purchased to control the trans-Siberian gas pipeline. The Soviets had obtained the system as part of a wide-ranging effort to covertly purchase or steal sensitive U.S. technology. The CIA reportedly found out about the program and decided to make it backfire with equipment that would pass Soviet inspection and then fail once in operation. The resulting event is reportedly the largest non-nuclear explosion in the planet's history.

1985-1987 -- Therac-25 medical accelerator. A radiation therapy device malfunctions and delivers lethal radiation doses at several medical facilities. Based upon a previous design, the Therac-25 was an "improved" therapy system that could deliver two different kinds of radiation: either a low-power electron beam (beta particles) or X-rays. The Therac-25's X-rays were generated by smashing high-power electrons into a metal target positioned between the electron gun and the patient. A second "improvement" was the replacement of the older Therac-20's electromechanical safety interlocks with software control, a decision made because software was perceived to be more reliable.

What engineers didn't know was that both the 20 and the 25 were built upon an operating system that had been kludged together by a programmer with no formal training. Because of a subtle bug called a "race condition," a quick-fingered typist could accidentally configure the Therac-25 so the electron beam would fire in high-power mode but with the metal X-ray target out of position. At least five patients die; others are seriously injured.

1988 -- Buffer overflow in Berkeley Unix finger daemon. The first internet worm (the so-called Morris Worm) infects between 2,000 and 6,000 computers in less than a day by taking advantage of a buffer overflow. The specific code is a function in the standard input/output library routine called gets() designed to get a line of text over the network. Unfortunately, gets() has no provision to limit its input, and an overly large input allows the worm to take over any machine to which it can connect.

Programmers respond by attempting to stamp out the gets() function in working code, but they refuse to remove it from the C programming language's standard input/output library, where it remains to this day.

1988-1996 -- Kerberos Random Number Generator. The authors of the Kerberos security system neglect to properly "seed" the program's random number generator with a truly random seed. As a result, for eight years it is possible to trivially break into any computer that relies on Kerberos for authentication. It is unknown if this bug was ever actually exploited.

January 15, 1990 -- AT&T Network Outage. A bug in a new release of the software that controls AT&T's #4ESS long distance switches causes these mammoth computers to crash when they receive a specific message from one of their neighboring machines -- a message that the neighbors send out when they recover from a crash.

One day a switch in New York crashes and reboots, causing its neighboring switches to [crash](http://www.cs.berkeley.edu/~nikitab/courses/cs294-8/hw1.html), then their neighbors' neighbors, and so on. Soon, 114 switches are crashing and rebooting every six seconds, leaving an estimated 60 thousand people without long distance service for nine hours. The fix: engineers load the previous software release.

1993 -- Intel Pentium floating point divide. A silicon error causes Intel's highly promoted Pentium chip to make mistakes when dividing floating-point numbers that occur within a specific range. For example, dividing 4195835.0/3145727.0 yields 1.33374 instead of 1.33382, an error of 0.006 percent. Although the bug affects few users, it becomes a public relations nightmare. With an estimated 3 million to 5 million defective chips in circulation, at first Intel only offers to replace Pentium chips for consumers who can prove that they need high accuracy; eventually the company relents and agrees to replace the chips for anyone who complains. The bug ultimately costs Intel $475 million.

1995/1996 -- The Ping of Death. A lack of sanity checks and error handling in the IP fragmentation reassembly code makes it possible to crash a wide variety of operating systems by sending a malformed "ping" packet from anywhere on the internet. Most obviously affected are computers running Windows, which lock up and display the so-called "blue screen of death" when they receive these packets. But the attack also affects many Macintosh and Unix systems as well.

June 4, 1996 -- Ariane 5 Flight 501. Working code for the Ariane 4 rocket is reused in the Ariane 5, but the Ariane 5's faster engines trigger a bug in an arithmetic routine inside the rocket's flight computer. The error is in the code that converts a 64-bit floating-point number to a 16-bit signed integer. The faster engines cause the 64-bit numbers to be larger in the Ariane 5 than in the Ariane 4, triggering an overflow condition that results in the flight computer crashing.

First Flight 501's backup computer crashes, followed 0.05 seconds later by a crash of the primary computer. As a result of these crashed computers, the rocket's primary processor overpowers the rocket's engines and causes the rocket to disintegrate 40 seconds after launch.

November 2000 -- National Cancer Institute, Panama City. In a series of accidents, therapy planning software created by Multidata Systems International, a U.S. firm, miscalculates the proper dosage of radiation for patients undergoing radiation therapy.

Multidata's software allows a radiation therapist to draw on a computer screen the placement of metal shields called "blocks" designed to protect healthy tissue from the radiation. But the software will only allow technicians to use four shielding blocks, and the Panamanian doctors wish to use five.

The doctors discover that they can trick the software by drawing all five blocks as a single large block with a hole in the middle. What the doctors don't realize is that the Multidata software gives different answers in this configuration depending on how the hole is drawn: draw it in one direction and the correct dose is calculated, draw in another direction and the software recommends twice the necessary exposure.

At least eight patients die, while another 20 receive overdoses likely to cause significant health problems. The physicians, who were legally required to double-check the computer's calculations by hand, are indicted for murder.

[**NASA’s Mars Climate Orbiter**](ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf) – On its mission to Mars in 1998 the Climate Orbiter spacecraft was ultimately lost in space. Although the failure bemused engineers for some time it was revealed that a sub contractor on the engineering team failed to make a simple conversion from English units to metric. An embarrassing lapse that sent the $125 million craft fatally close to Mars’ surface after attempting to stablize its orbit too low. Flight controllers believe the spacecraft ploughed into Mars’ atmosphere where the associated stresses crippled its communications, leaving it hurtling on through space in an orbit around the sun.

[**Ariane 5 Flight 501**](https://en.wikipedia.org/wiki/Cluster_(spacecraft)#Launch_failure)**–**Europe’s newest un-manned satellite-launching rocket reused working software from its predecessor, the Ariane 4. Unfortunately, the Ariane 5’s faster engines exploited a bug that was not found in previous models. Thirty-six seconds into its maiden launch the rocket’s engineers hit the self destruct button following multiple computer failures. In essence, the software had tried to cram a 64-bit number into a 16-bit space. The resulting overflow conditions crashed both the primary and backup computers (which were both running the exact same software). The Ariane 5 had cost nearly $8 billion to develop, and was carrying a $500 million satellite payload when it exploded. In the video below you can see the engineer struggle to comprehend what he’s just seen on his screen as the rocket explodes before calmly writing down a short note, probably the letters F…A…I….L.

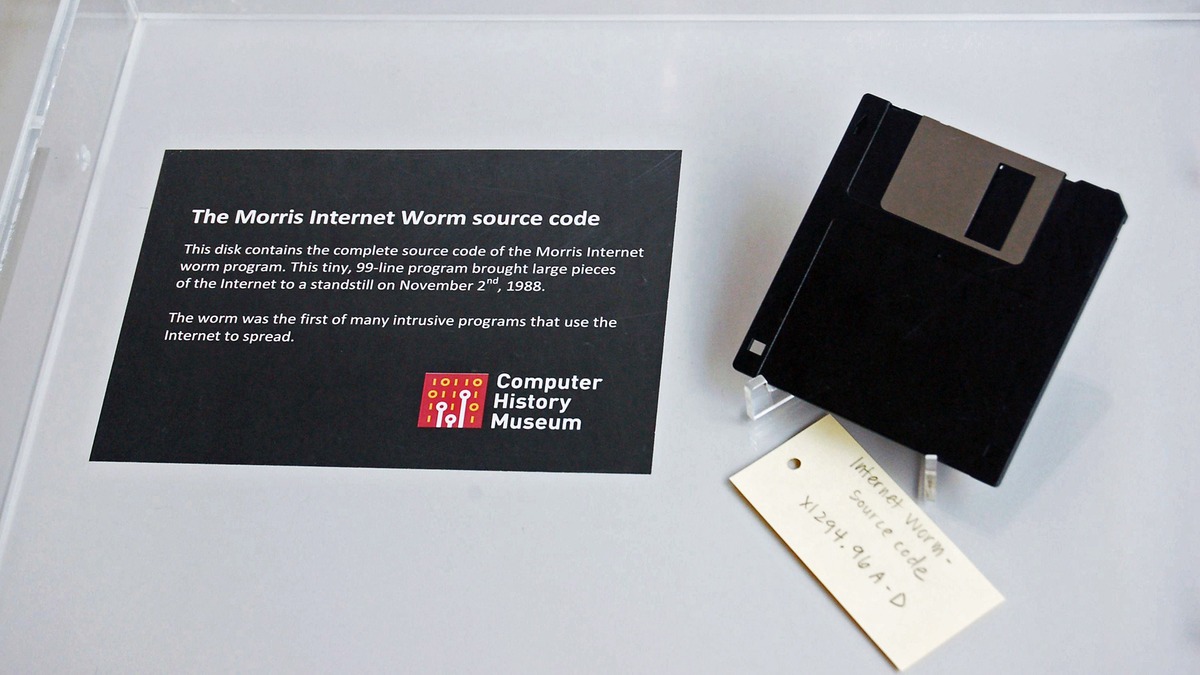
[**EDS Child Support System**](http://news.bbc.co.uk/2/hi/uk_news/3235394.stm)**–**In 2004, EDS introduced a highly complex IT system to the U.K.’s Child Support Agency (CSA). At the exact same time, the Department for Work and Pensions (DWP) decided to restructure the entire agency. The two pieces of software were completely incompatible, and irreversible errors were introduced as a result. The system somehow managed to overpay 1.9 million people, underpay another 700,000, had $7 billion in uncollected child support payments, a backlog of 239,000 cases, 36,000 new cases “stuck” in the system, and has cost the UK taxpayers over $1 billion to date.

[**Soviet Gas Pipeline Explosion**](http://en.wikipedia.org/wiki/Siberian_pipeline_sabotage)– The Soviet pipeline had a level of complexity that would require advanced automated control software. The CIA was tipped off to the Soviet intentions to steal the control system’s plans. Working with the Canadian firm that designed the pipeline control software, the CIA had the designers deliberately create flaws in the programming so that the Soviets would receive a compromised program. It is claimed that in June 1982, flaws in the stolen software led to a massive explosion along part of the pipeline, causing the largest non-nuclear explosion in the planet’s history.

[**Heathrow Terminal 5 Opening**](http://news.bbc.co.uk/2/hi/uk_news/7314816.stm)**–** Just before the opening of Heathrow’s Terminal 5 in the UK, staff tested the brand new baggage handling system built to carry the vast amounts of luggage checked in each day. Engineers tested the system thoroughly before opening the Terminal to the public with over 12,000 test pieces of luggage. It worked flawlessly on all test runs only to find on the Terminal’s opening day the system simply could not cope. It is thought that ‘real life’ scenarios such as removing a bag from the system manually when a passenger had left an important item in their luggage, had caused the entire system to become confused and shut down. Over the following 10 days some 42,000 bags failed to travel with their owners, and over 500 flights were cancelled.

[**The Mariner 1 Spacecraft**](http://edn.com/electronics-blogs/edn-moments/4418667/Mariner-1-destroyed-due-to-code-error--July-22--1962) – On a mission to fly-by Venus in 1962, this spaceship barely made it out of Cape Canaveral when a software-coding error caused the rocket to veer dangerously off-course, threatening to crash back to earth. Alarmed, NASA engineers on the ground issued a self-destruct command. A review board later determined that the omission of a hyphen in coded computer instructions allowed the transmission of incorrect guidance signals to the spacecraft. The cost for the rocket was reportedly more than $18 million at the time.

[**The Morris Worm**](http://en.wikipedia.org/wiki/Robert_Tappan_Morris#The_Morris_worm) – A program developed by a Cornell University student for what he said was supposed to be a harmless experiment wound up spreading wildly and crashing thousands of computers in 1988 because of a coding error. It was the first widespread worm attack on the fledgling Internet. The graduate student, Robert Tappan Morris, was convicted of a criminal hacking offense and fined $10,000. Morris’s lawyer claimed at the trial that his client’s program helped improve computer security. Costs for cleaning up the mess may have gone as high as $100 Million. Morris, who interestingly co-founded the startup incubator Y Combinator, is now a professor at the Massachusetts Institute of Technology. A disk with the worm’s source code is now housed at the University of Boston.

[](https://raygun.io/blog/wp-content/uploads/2014/05/Morris-Worm.jpg)

[**Patriot Missile Error**](http://www.ima.umn.edu/~arnold/disasters/patriot.html)**–**Sometimes, the cost of a software glitch can’t be measured in dollars. In February of 1991, a U.S. Patriot missile defence system in Saudi Arabia, failed to detect an attack on an Army barracks. A government report found that a software problem led to an “inaccurate tracking calculation that became worse the longer the system operated.” On the day of the incident, the system had been operating for more than 100 hours, and the inaccuracy was serious enough to cause the system to look in the wrong place for the incoming missile. The attack killed 28 American soldiers. Prior to the incident, Army officials had fixed the software to improve the Patriot system’s accuracy. That modified software reached the base the day after the attack.

[**Pentium FDIV bug**](http://en.wikipedia.org/wiki/Pentium_FDIV_bug) – When a math professor discovered and publicized a flaw in Intel’s popular Pentium processor in 1994, the company’s response was to replace chips upon request to users who could prove they were affected. Intel calculated that the error caused by the flaw would happen so rarely that the vast majority of users wouldn’t notice. Angry customers demanded a replacement for anyone who asked, and Intel agreed. The episode cost Intel $475 million.

[**Knight’s $440 Million Error**](http://www.businessweek.com/articles/2012-08-02/knight-shows-how-to-lose-440-million-in-30-minutes)– One of the biggest American market makers for stocks struggled to stay afloat after a software bug triggered a $440 million loss in just 30 minutes. The firm’s shares lost 75 percent in two days after the faulty software flooded the market with unintended trades. One of Knight’s trading algorithms reportedly started pushing erratic trades through on nearly 150 different stocks, sending them into spasms.

**Honourable mention:**[**NOAA-19 Satellite**](http://www.nasa.gov/pdf/65776main_noaa_np_mishap.pdf)**–**Although not a software error, on September 6, 2003, the satellite was badly damaged while being worked on at the Lockheed Martin Space Systems factory. The satellite fell to the floor as a team was turning it to a horizontal position. An inquiry into the mishap determined that it was caused by a lack of procedural discipline throughout the facility. While the turn-over cart used during the procedure was in storage, a technician removed twenty-four bolts securing an adapter plate to it without documenting the action. The team subsequently using the cart to turn the satellite failed to check the bolts, as specified in the procedure, before attempting to move the satellite. Repairs to the satellite cost $135 million.

[](https://raygun.io/blog/wp-content/uploads/2014/05/nasa.jpg)