Softwarefehler (gesammelte Notizen)

Der erste Computerfehler

http://www.dpunkt.de/leseproben/3-88229-198-2/Kapitel_1.pdf

Unter Computer-Freaks werden Software-Schwachstellen gern mit dem englischen Begriff "bug" bezeichnet, was auf Deutsch soviel wie "Käfer" oder "Insekt" heißt. Der Legende nach geht dies auf ein Ereignis aus dem Jahr 1945 zurück:

Im US-Marine-Waffenzentrum in Dahlgren, Virginia, stand damals einer der ersten Großrechner der Welt, der Harvard Mark II. Dieser Urvater heutiger PCs arbeitete noch mit mechanischen Relais. Eines Tages wurde die Maschine lahm gelegt - und zwar durch eine Motte, die in einem der Schalter festgeklemmt war. Die Computer-Wissenschaftlerin Grace Hopper entfernte das Insekt und klebte es in ihr tägliches Fehler-Logbuch ein - damit war das erste historisch dokumentierte "debugging" eines Systems durchgeführt. Fehlerreport samt Motte sind übrigens noch heute im Smithsonian Institute zu besichtigen.

Im September 1999 scheiterte die viel beachtete Mission der amerikanischen Mars-Sonde "Climate Orbiter" spektakulär - Hauptursache war offenbar ein simpler Umrechnungsfehler in der Software des Raumschiffs: Während ein NASA-Entwicklerteam für sein Navigationsprogramm das metrische System (Zentimeter, Meter usw.) benutzte, basierte ein zweites seine Steuerroutinen auf den britischen imperialen Maßeinheiten wie Inch oder Fuß.

Statt in eine Umlaufbahn um den roten Planeten einzuschwenken, geriet die Sonde durch den Rechenfehler zu nah an die Atmosphäre und verglühte dort. Die 125 Millionen Dollar teure NASA-Mission löste sich nach zehnmonatigem Flug so buchstäblich in Nichts auf.

Die Normalität von Bugs

Wie die bisherigen Beispiele zeigen, sind Bugs in ihren verschiedensten Ausprägungen eher die Regel denn die Ausnahme. Statistiken von Hard- und Softwareherstellern bestätigen dies eindrucksvoll.

Prozessorhersteller Intel etwa schätzt, dass in der Systemarchitektur seines Pentium-Prozessors etwa 80 bis 90 Bugs versteckt seien - es braucht hier also nichtmal ein "echtes" Insekt, um Schaltkreise aus der Bahn zu werfen.

Normale Software enthält zirka 25 Fehler pro 1000 Codezeilen, als "gut" gilt ein Programm, wenn maximal zwei Fehler in 1000 Zeilen vorkommen. Die NASA gibt als Maßstab für ihre Space-Shuttle-Software weniger als einen Fehler pro 10.000 Zeilen vor. Das heimische Betriebssytem kann da natürlich nicht mithalten: Windows 95 etwa umfasste etwa 10 Millionen Zeilen Programmcode mit bis zu 200.000 Fehlern.

Der berühmteste Bug

Der berühmteste - und vielleicht auch am meisten überschätzte - Fehler ist der "Y2K" (Year 2000 - Jahr 2000)-Bug. Er betraf die Zeitumstellung auf Jahresangaben mit "20.." und sorgte 1999 für riesiges Aufsehen.

Technischer Hintergrund: Die ersten Softwareprogramme waren aus Speicherplatzmangel so geschrieben, dass sie die Jahreszahl nur zweistellig speicherten. Sobald sie auf eine "00" stießen, wurde diese als das Jahr 1900 interpretiert.

Ohne große Fantasie kann man sich vorstellen, dass ein solcher Fehler etwa im Finanzbereich zu schweren Fehlkalkulationen führen kann. Das Problem war schon Ende der 70er Jahre bekannt. Damals gingen die Programmierer jedoch davon aus, dass ihre Software ohnehin nicht bis zum Jahr 2000 eingesetzt werden würde. Eine Fehleinschätzung, denn 1999 mussten sich dann Horden von IT-Experten daran machen, die Nachlässigkeiten ihrer Vorgänger zu korrigieren. Man schätzt, dass weltweit etwa 600 Milliarden Dollar für das Y2K-Debugging ausgegeben wurden.

Wieviel diese Rettungsmaßnahmen tatsächlich brachten, weiß niemand - jedenfalls kam und ging das Jahr 2000, ohne dass Y2K die Weltbörse zum Ein- oder Flugzeuge zum Absturz gebracht hätte. Den Millennium-Bug muss man vielmehr als kulturelles Phänomen sehen, passte er doch 1999 perfekt in die Horrorszenarien von Weltuntergangspropheten und Endzeitfantasten.

Der schlimmste Bug

Den wirklich allerschlimmsten Bug kennt jeder PC-Anwender wohl am besten - es ist nämlich derjenige, der vor dem Rechner sitzt: Sie selbst. Nicht nur, dass fehlerbehaftete Programme letztendlich von Menschen geschrieben wurden, auch in der Bedienung des Computers macht jeder von uns tagtäglich Dutzende Fehler, die selbst von der ausgeklügeltsten Software nicht abgefangen werden können. Was aber weiter nicht tragisch ist - denn wie gesagt, Computer sind auch nur Menschen.

History's Worst Software Bugs

http://www.wired.com/software/coolapps/news/2005/11/69355?currentPage=all

http://able2know.org/topic/129489-1

Simson Garfinkel ✉ 11.08.05

Last month automaker Toyota announced a recall of 160,000 of its Prius hybrid vehicles following reports of vehicle warning lights illuminating for no reason, and cars' gasoline engines stalling unexpectedly. But unlike the large-scale auto recalls of years past, the root of the Prius issue wasn't a hardware problem -- it was a programming error in the smart car's embedded code. The Prius had a software bug.

With that recall, the Prius joined the ranks of the buggy computer -- a club that began in 1945 when engineers found a moth in Panel F, Relay #70 of the Harvard Mark II system.The computer was running a test of its multiplier and adder when the engineers noticed something was wrong. The moth was trapped, removed and taped into the computer's logbook with the words: "first actual case of a bug being found."

Sixty years later, computer bugs are still with us, and show no sign of going extinct. As the line between software and hardware blurs, coding errors are increasingly playing tricks on our daily lives. Bugs don't just inhabit our operating systems and applications -- today they lurk within our cell phones and our pacemakers, our power plants and medical equipment. And now, in our cars.

But which are the worst?

It's all too easy to come up with a list of bugs that have wreaked havoc. It's harder to rate their severity. Which is worse -- a security vulnerability that's exploited by a computer worm to shut down the internet for a few days or a typo that triggers a day-long crash of the nation's phone system? The answer depends on whether you want to make a phone call or check your e-mail.

Many people believe the worst bugs are those that cause fatalities. To be sure, there haven't been many, but cases like the Therac-25 are widely seen as warnings against the widespread deployment of software in safety critical applications. Experts who study such systems, though, warn that even though the software might kill a few people, focusing on these fatalities risks inhibiting the migration of technology into areas where smarter processing is sorely needed. In the end, they say, the lack of software might kill more people than the inevitable bugs.

What seems certain is that bugs are here to stay. Here, in chronological order, is the *Wired News* list of the 10 worst software bugs of all time ... so far.

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.

Während des Falkland-Kriegs 1982 wurde eine britische Fregatte durch zwei EXOCET-Raketen zerstört: Das moderne, computergesteuerten Anti-Raketen-System der Fregatte hatte die beiden von argentinischer Seite abgefeuerten Geschosse auch rechtzeitig erkannt. Da es sich bei den EXOCETs aber um französische Produkte (NATO-Gerät also) handelte, hatte das System die Raketen als 'befreundet' eingestuft, so dass ihnen gestattet wurde, in den Abwehrkreis der Fregatte einzufliegen: ein Softwarefehler.
Zwei britische Soldaten kamen ums Leben.

1982 -- Soviet gas pipeline. Operatives working for the Central Intelligence Agency allegedly (.pdf) 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.

#1 World War III... Almost (1983)
In 1983, Soviet early warning satellites picked up sunlight reflections off cloud-tops and mistakenly interpreted them as missile launches in the United States. Software was in place to filter out false missile detections of this very nature, but a bug in the software let the alerts through anyway. The Russian system instantly sent priority messages up saying that the United States had launched five ballistic missiles. Protocol in such an event was to respond decisively, launching the entire soviet nuclear arsenal before any US missile detonations could disable their response capability. The duty officer for the system, one Lt Col Stanislav Petrov, intercepted the messages and flagged them as faulty, stopping the near-apocalypse. He claimed that he had a “funny feeling in my gut” about the attack, and reasoned if the U.S. was really attacking they would launch more than five missiles.
Result - Thankfully nothing. However, the world was literally minutes away from “Global Thermal Nuclear War”. **Any retaliatory missile launched by the Soviets would have triggered a like response from the U.S., eventually leading to a total launch of all systems from both sides.**

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, 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.

Im November 1988 führe ein Programmierfehler dazu, dass ein vom Studenten Robert Tappan Morris geschriebenes Programm, das sich unschädlich durchs Internet bewegen sollte, in kürzester Zeit etwa 6.000 Rechner am Netz befiel und sie durch Überlastung lahm legte – ungefähr 10 Prozent des damals bestehenden Internets. Kurioserweise war der Vater von Morris war zur selben Zeit Leiter des Zentrums für Computersicherheit des Geheimdienstes NSA (National Security Agency). **Morris Jr. wurde im Januar 1990 zu einer Bewährungsstrafe, 400 Stunden Sozialarbeit und 10.000 Dollar Geldstrafe verurteilt.**

Am Schwarzen Montag, dem **19. Oktober 1987**, führten Trading-Programme zu einem Crash der New Yorker Börse. Der Dow-Jones-Index verlor an diesem Tag mehr als 20 Prozent. (Note: Damals wurde erst etwa jeder fünfte Aktienhandel von einem Computer entschieden – in 2011 aber kämpfen die Betreiber entsprechender Systeme darum, dass die Glasfaserkabel zwischen ihrem System und dem der Börse so kurz wie nur irgend möglich sind: es geht hier um Meter, nicht etwa um Kilometer (!)).

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.

Zwischen 1993 und 1994 gab es mehrere Versuche, den Flughafen in Denver neu zu eröffnen. Auf dem supermodernen Flughafen sollte ein vollautomatisches Gepäcksystem im Zusammenspiel von 150 Computern, Laserscannern und Fotozellen zum Einsatz kommen. Ergebnis: Gepäckstücke wurden auf Wagen geworfen, die nicht da waren, Wagen fielen aus den Fahrspuren heraus, Koffer wurden zerquetscht und gingen verloren – das Gepäck wurde wieder von Hand sortiert. **Die Verluste von täglich 1,1 Millionen Dollar durch die**

fehlerhafte Anlage summierten sich am Ende auf 3,2 Milliarden Dollar. (Quelle: http://www.channelpartner.de/index.cfm?pid=53&pk=277097&p=2)

1995 sollte das von 50 Eisenbahnern betriebene alte Stellwerk des Bahnhofs Hamburg-Altona durch ein digitales System, zu dessen Betrieb nur noch 10 Personen nötig sind, ersetzt werden. Nach Inbetriebnahme am 13. März stürzten alle zehn Minuten die Rechner ab. Die Schließung des Stellwerks erzeugte ein **europaweites Verkehrschaos im Bahnverkehr**. Es dauerte zwei Tage, bis der Fehler gefunden war: Ein Zwischenspeicher für die Stellbefehle im Hauptrechner der neuen Anlage war zu klein ausgelegt. Dadurch kam es zu einem Datenüberlauf, infolgedessen der Rechner sich – und damit das ganze Stellwerk – abschaltete.

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.**

Alleine in den Jahren **2005 bis 2009** verzeichnete die amerikanische Food and Drug Administration (FDA) mehr als **56.000 Berichte über Probleme mit Infusionspumpen**. Probleme, die schwerwiegende Konsequenzen für die Patienten hatten: von simplen Fehlfunktionen (62%) über Verletzungen (34%) bis hin zu Todesfällen (etwa 1%). **Zu den häufigsten Ursachen gehörten Software-Fehler**, aber auch Probleme mit der Gebrauchstauglichkeit der Geräte. Viele dieser Fehler erwiesen sich im Nachhinein als absehbar und hätten daher eigentlich im Vorfeld verhindert werden können.

In 2008 war ein Airbus 330 auf dem Flug von Singapur nach Perth in einer Höhe von 11277 Metern plötzlich um insgesamt 323 Meter abgestürtzt. 110 der 303 Fluggäste und 12 Mitglieder der Besatzung wurden verletzt. Ursache war, so der Bericht des Australian Transport Saftey Bureau, ein Fehler im Computersystem: in einer der insgesamt drei Air Data Inertial Reference Units (Adiru). Die Adiru ist ein Gerät, das wichtige Flugdaten an das Electronic Flight Instrument System und andere Systeme liefert. Nach Auskunft der Verkehrsbehörde war die Wahrscheinlichkeit, dass passieren konnte, was tatsächlich geschah, äußerst gering: In 128 Millionen Stunden hätten baugleiche Adiru-Typen nur drei Ausfälle der beschriebenen Art herbeiführen können.

Temporärer Ausfall des Reaktorüberwachungssystems: Wie die japanische Atomenergie-Sicherheitsbehörde NISA am Samstag (den 31.12.2011) mitteilte, war es am Freitag zu einem Ausfall des Computersystems zur Echtzeit-Überwachung des Zustandes von Atomreaktoren ***im ganzen Land*** gekommen. Für ***26 Stunden*** habe man keine Daten des "Emergency Response Support System" (ERSS) erhalten. Der Fehler wird auf einen **Softwarefehler** zurückgeführt, so Sprecher der NISA. (Quelle: http://www.spreadnews.de/japan-aktuell-ausfall-des-akw-uberwachungssystems-der-nisa/1119330/)

Spectacular software failures

http://www.bizcommunity.com/Article/196/16/68783.html

20 Dec 2011 11:27Submit a commentBizLike

Consultants from Software Quality Systems (SQS) voted on software failures, which hit the headlines and, in their opinion, made a big impact in terms of money lost ($) or inconvenience caused.

Phil Codd

Phil Codd, MD & chief markets officer (Northern Europe, India & South Africa) at SQS comments, "Our top ten list of 2011 shows that software failures are costing companies and consumers large amounts of money. What is worse is that people are losing jobs and in some cases their liberty because of avoidable software failures.

"The main problem caused by software bugs is negative financial impact and, in almost every case, consumers end up losing out. Deficiencies in software quality often result in costly emergency fixes and/or damage to a brand's reputation, but each of our top ten 2011 software failure examples could easily have been avoided through an effective quality management strategy identifying and

resolving potential glitches before they appear," he concluded.

Top 10 Software Failures in 2011:

1. **Financial services giant fined $25 million for hiding software glitch that cost investors $217 million** - $ and ?. A software error in the investment model used to manage client assets resulted in this international financial services giant being fined $25 million (£15.7 million) by the US Securities and Exchange Commission (SEC). The company also had to repay the $217 million (£136 million) backers lost when told that market volatility rather than software failure was to blame for their investment losses.
2. **Computer system bugs cause Asian banking facilities' downtime** - ?. Computer system problems at one of Japan's largest banks resulted in a nationwide ATM network of more than 5600 machines going offline for 24 hours, internet banking services being shut down for three days, delays in salary payments worth $1.5 billion (£939 million) into the accounts of 620,000 people and a backlog of more than 1 million unprocessed payments worth around $9 billion (£5.64 billion).
3. **Cash machine bug benefits customers by giving them extra money** - $. An Australian bank began giving out large sums of money from 40 cash machines across one city. Officials at the company said they were operating in stand-by mode, so could not identify the account balances of customers.
4. **Leading smartphones suffer an international blackout** - ?. Core and back-up switch failures resulted in network services across Europe, the Middle East, Africa and Latin America going down for three to four days. The blackout left millions without email, web browsing or instant messaging services and were reportedly due to server problems at one data centre, in Slough.
5. **Bugs in social networking app for tablet just hours after delayed release** - ?. Just hours after its release, this social networking sites' long-awaited tablet app was already receiving reports about minor bugs from clicking through to pages via panel icons to problems posting comments.
6. **22 people wrongly arrested in Australia due to failures in new NZ $54.5 million courts computer system** - $ and ?. A new NZ $54.5 million ($42.7 million or £26.8 million) computer system linking New South Wales courts and allowing documents to be lodged electronically led to damages claims for unlawful arrest and malicious prosecution, after 3600 defects in the electronic transfer of data from the courts to the police's database led to the wrongful arrest of 22 individuals.
7. **50 500 cars recalled after airbag-related software glitch** - $ and ?. A glitch in the automaker's software design and testing approach, that meant airbags for passengers in the right rear seat during a crash may not be deployed, resulted in the recall of 47 401 vehicles in the US and a further 3 099 in Canada and Mexico.
8. **Recall of one million cars addresses fire and rollaway concerns** - $ and ?. A Japanese car company was forced to initiate a worldwide recall of over one million vehicles affected by a design flaw allowing residue from window cleaners to accumulate, which can degrade the switch's electrical contacts and potentially cause a fire over time. This recall followed a global 2.5 million recall by the same company due to design flaws that allowed vehicles to shift out of park and engine stalls.
9. **Telecoms glitch affects 47 000 customers' meter readings and costs company NZ $2.7 million** - $ and ?. After a software glitch that resulted in customers hitting their data limits early, some 47 000 customers, who were overcharged, were reimbursed by a

New Zealand telecoms company in a NZ $2.7 million ($2.1 million or £1.3 million) payout.

10. **Army computer glitches hinder co-ordinated efforts in insurgent tracking**. An army computing system designed to share real-time intelligence with troops on the front line has hindered troops by being unable to perform simple analytical tasks. The $2.7 billion cloud-based computing network system runs slowly when multiple users are on the system at the same time and the system's search tool made finding the reports difficult as the information mapping software was not compatible with the army's existing search software.

$ = US Dollars unless otherwise stated and ? = stress.

http://able2know.org/topic/129489-1

Honorable Mention #1 - **LA Airport Flights Grounded (2007)**
A single faulty piece of embedded software, on a network card, sends out faulty data on the United States Customs and Border Protection network, bringing the entire system to a halt. Nobody is able to leave or enter the U.S. from the LA Airport for over eight hours.
Result - Over 17,000 planes grounded for the duration of the outage

http://www.benmeadowcroft.com/reports/systemfailure/

As a cause of system failure, poor development practices are one of the most significant. This is due to the complex nature of modern software.

An example of poor development practices causing a system failure can be found in the experience of the Pentagon's National Reconnaissance Office (NRO). The inadequate testing of the delivery system of Titan IV rocket. Two Titan rockets were lost, meaning that expensive military equipment necessary to the U.S. Governments defence program (namely early warning satellites) were unable to be deployed. The head of the N.R.O. has attributed this error to "a misplaced decimal point" in software, which controlled the rocket. See http://spacer.com/spacecast/news/launchers-99l.html

Observations on:

How buggy or risky is Operating Systems Software?

In 1998, MS website claimed NT5 would fix more than 10.000 outstanding bugs in NT4. We conclude: NT4 certainly contained much more than only 10.000 serious bugs.

http://www.zdnet.co.uk/news/it-strategy/2000/02/14/bugfest-win2000-has-63000-defects-2076967/

According to a Microsoft-internal memo leaked to the press at the time when Win2000 was released, Microsoft's Prefix tool discovered that the final Win2000 release code still had more than **63,000 "potential issues"** that could emerge as problems.

Microsoft estimated that **28,000 of these were likely to be "real" problems**.

It is well known that Win95 code was around 10 Mio LOC, so if we assume that Win2000 had around 28 Mio LOC, there was likely to be ***one problem per 1000 LOC in Win2000***.

Despite of this, as we all know, Win2000 was a very usable software package.

On the other hand (see above), most simple bugs can create a lot of damage: Because of "a misplaced decimal point" in the code for the delivery system of the Titan IV rocket, the U.S. Government lost two of these quite expensive rockets. Without them the military could not deploy early warning satellites.

Trends wurden noch kaum untersucht:

http://www.greiterweb.de/spw/FehlerReduzierungswege.htm