**What is 'Software Quality Assurance'?**  
Software QA involves the entire software development PROCESS - monitoring and improving the process, making sure that any agreed-upon processes, standards and procedures are followed, and ensuring that problems are found and dealt with. It is oriented to 'prevention'. (See the [Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA' category for a list of useful books on Software Quality Assurance.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is 'Software Testing'?**  
Testing involves operation of a system or application under controlled conditions and evaluating the results (e.g., 'if the user is in interface A of the application while using hardware B, and does C, then D should happen'). The controlled conditions should include both normal and abnormal conditions. Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should. It is oriented to 'detection'. (See the [Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software Testing' category for a list of useful books on Software Testing.)

* Organizations vary considerably in how they assign responsibility for QA and testing. Sometimes they're the combined responsibility of one group or individual. Also common are project teams and agile teams that include a mix of testers and developers who work closely together, with overall testing and QA processes monitored by a project manager, a scrum master, or other appropriate person. It will depend on what best fits an organization's size, development approach, and business structure.
* Note that testing can be done by machines or people. When done by machines (computers usually) it's often called 'automated testing' - see [the SoftwareQATest.com LFAQ page](http://www.softwareqatest.com/qat_lfaq1.html#LFAQ1_8) for more information about automated testing. Of course a human still has to develop the automation strategy and test cases and write test automation code.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What are some recent major computer system failures caused by software bugs?**

* A bug in a recent version of a major operating system enabled anyone to create a root acount in certain situations, according to news stories in November of 2017. A security patch was quickly released, which reportedly also introduced a new bug, though one of much less severity. The company issued an apology and indicated they would examine their development processes.
* Media reports in September 2017 indicated that a large number of airlines worldwide were experiencing flight check-in system problems for a period of hours, resulting in delays and long lines. Reportedly the problem was in a computer system used by more than 100 airlines around the world, and also used by hotels, tour operators, insurers, car rental agencies, passenger railroads, cruise lines, travel agencies and individual travellers.
* In May 2017 it was reported that a major electronic health record software vendor was fined more than $150 million for concealing that their software did not meet testing certification requirements; the false certification had resulted in false claims for government incentive payements. One of the software developers involved was individually fined $50,000 and $30 million was awarded to a whisteleblower who first reported the issue.
* In April and May of 2017 news reports indicated various computer system failures occurred for each of several major European airlines, resulting in grounded airplanes, cancelled flights, long lines at airports, and outages at call centers and web sites. Several articles appeared analyzing the reasons behind the numerous and continuing system failures at major airlines, with the root causes being attributed to human error and system complexity.
* Ongoing reports of problems with a major U.S. government IT project were continuing in April of 2017. The project, which began in 2005, had a goal of digitizing manual paper-based processing, including background checks, for more than 90 different forms. Reportedly as of 2017, digitization of processing for only two of the forms had been implemented, the project was billions of dollars over budget and years behind schedule, and what was implementations had numerous outages, errors, and security issues and required manual interventions to complete processing.
* A nationwide (U.S.) 5-hour outage of the 911 emergency calling system utilized by customers of a major telecom was reported in March of 2017. It affected more than 12,000 emergency calls during the outage.
* News reports in February 2017 described a 5-hour outage of one of the most heavily used regions of a major internet cloud service, impacting many popular sites/apps/publications/companies. Reportedly these included Github, Medium, Slack, Coursera, Bitbucket, Citrix, Expedia, Flipboard, Yahoo! Mail, Netflix, Tinder, Airbnb, Reddit, IMDb, Business Insider, SiriusXM, image availability for many publications, many IoT security cameras and apps, many IoT thermostats, and other IoT hardware. The cloud service was reportedly unable to update its own service status reporting web site for several hours initially.
* Several major airlines suffered various significant computer system problems during the period July-October 2016, resulting in thousands of flight delays or cancellations worldwide. Among other impacts, this led to a U.S. Congressional inquiry as to why airline computer systems had become so prone to failure.
* The European Space Agency's ExoMars Schiaparelli spacecraft crash landed on Mars in October 2016 as a result of problems in handling a small amount of bad sensor data in the spacecraft's computer systems. It is believed that a software fix rather than a more difficult hardware fix will resolve the problem for future missions.
* A September 2016 update of a major smartphone OS resulted in many users' loss of use of their smartphones. A series of bug-fix releases over the succeeding months resolved many issues, but sometimes introduced additional issues.
* In January 2016 there were news reports that a major airline had to ground all US flights due to computer failure, resulting in delays for more than 25% of it's flights on the day of the failure. The issues were resolved later that same day. A week later it was reported that system failures at another major airline brought down the airline's web site, mobile app, terminal information screens and reservation desk system. Many flights were delayed and more than 100 flights were cancelled.
* The computer systems handling passenger service for a major North American airline failed in October of 2015 according to news reports. It resulted in delays for more than 20% of all the airlines' flights, extremely long lines at checkin, and manual handling of tickets and flight boarding for most of a day. The cause was not reported but was speculated to be overloading of complex legacy systems.
* A major subway system's new rail cars had to be removed from service during August of 2015 due to a software problem; the rail cars had been put into service but during use had to be halted and passengers offloaded; the rail cars were to remain out of service until a software update was available.
* Problems in the air traffic control systems in the eastern U.S. in August of 2015 resulted in the delay of more than 3000 flights and more than 600 cancelled flights. For a short period there were almost no airplanes in the skies over the mid-Atlantic region of the country. Reportedly the cause was problems with the computer system that processed flight plans at a major air traffic control center.
* A major worldwide provider of news and data for financial institutions and investors was unavailable for several hours during the trading day in April 2015, resulting in a halt to trading activities at many institutions. The cause was attributed to multiple simultaneous computer system failures.
* Bugs in the computer system of a major urban police department were reported to have compromised potentially thousands of criminal cases over a period of years. News reports of March 2015 indicated that an extensive review of past criminal cases was under way to determine which cases had been affected.
* In February of 2015 it was reported that an entire nation's air traffic control system crashed due to a bug in a single line of code (among the millions of lines of code in the air traffic control systems). The system was safely fixed within an hour, however thousands of travelers were left grounded and had flights delayed.
* One of the major operating systems was found to have a bug that had been in existence for at least 19 years, according to reports in November of 2014. The critical security flaw potentially allowed remote control of a user's computer by hackers. The flaw was patched by the time of the announcement.
* A bug-fix upgrade to another major operating system was pulled back within a few hours of it's release in September of 2014 after a large number of reports of new significant bugs. The company apologized and released another new upgrade a day later.
* In July of 2014 software problems with a nationwide U.S. professional exam app resulted in failed or delayed online submissions of exam answers to the exam management service. The exam submission deadlines had to be extended to allow for eventual processing. The exam management company issued an apology.
* After spending $130 million on it's problematical health insurance exchange, one of the 14 U.S. states that opted to create their own health insurance exchange (rather than utilize a federal-government-provided exchange) hired a new contractor in April 2014 to redo the site. Among the many problems with the initial site since it went live in October 2013, reportedly hundreds of enrollees received enrollment information with names and birth dates of other enrollees. It was estimated the revamping would cost another $60 million. Additionally, the primary contractor and subcontractor for the site were embroiled in a lawsuit with one another and at last report were in arbitration. Eventually it was reported that the prime contractor agreed to pay $45 million to the government to avoid a lawsuit.
* In April of 2014 the 911 emergency calling system for 7 U.S. states was reportedly unavailable for 6 hours due to a software bug resulting in more than 6000 unhandled emergency calls.
* A large number of reports and discussions appeared in the media in Feb 2014 concerning bugs in a popular decentralized digital currency. Although a major digital currency exchange blamed certain of the bugs as a cause of a major monetary loss equivalent to hundreds of millions of dollars, there was considerable controversy as to the significance of the bugs in contributing to any losses. Although the problematical exchange shut down, other exchanges remained open and the digital currency remains popular.
* A major automobile manufacturer recalled nearly 2 million vehicles in February 2014 to fix a software problem that could cause problems in the vehicle's electronics or could cause it to partially shut down.
* The timed online entrance exam for one of the most selective technology magnet high schools in the U.S. experienced system problems including frozen screens and lost essays in January 2014. Afterwards school officials were assessing the situation to determine how to deal with the many students whose applications were blocked or disadvantaged because of the problems.
* In January 2014 a major free email service failed, along with many of the company's other popular services, due to a software bug, resulting in service outages for millions of users. The company was able to resolve the problem for most users in under an hour and issued an apology.
* Widespread reports appeared in the media in October 2013 about significant bugs in an online university application web site used by students to apply to one or more of hundreds of universities in the U.S. and several other countries. There were reports of uploading problems, loss of parts or all of required essays, problems with formatting, problems with recommendation letters, and more. Some colleges offered to extend their application deadlines to help mitigate the problem.
* In October of 2013 the U.S. federal government opened a new health insurance exchange web site that, during its first few months of operation, generated major national and worldwide press coverage of its many reported problems. The problems were attributed to, among other things, inadequate time allowed for system testing. A well-publicized 'tech surge' was initiated to attempt to improve the site.
* A major Asian stock market was whipsawed on a day in August 2013, reportedly due to bugs in an Asian brokerage's securities order system which resulted in more than $3 billion of incorrect trading orders. It was also reported that it caused a loss of $32 million to the brokerage, a significant fall in its stock price, and restrictions and investigations by the country's regulatory agency.
* During a short period in the latter half of August 2013 a diverse variety of major businesses in categories such as media, cloud services, email, stock markets, search engines, online retail, and investment banking suffered online outages and disruptions reportedly due to software problems, network problems, or unknown/unreported causes. During one set of related outages it was reported that worldwide internet traffic dropped 40%.
* A software bug in the trading system of a major investment bank was reported to have caused a large percentage of erroneous derivatives trades during the first 15 minutes of the trading day on a major securities exchange in August 2013. Exchanges worked through the day to determine which trades had to be cancelled.
* In April 2013 it was reported that a major financial exchange was unable to open for trading due to a software glitch. Once fixes were in place trading resumed 3 hours late.
* Hundreds of computer-controlled jail cell locks were unexpectedly opened at a 1000-inmate prison in April 2013 due to what was believed to be a software problem, according to media reports. A security emergency was declared and no inmates escaped. It was the second such incident within a week. At last report the systems were still being tested to determine the cause of the malfunction.
* In February 2013 a mobile device manufacturer reached a settlement agreement with the U.S. government because, among other things, it "failed to provide its engineering staff with adequate security training, failed to review or test the software on its mobile devices for potential security vulnerabilities...". The company agreed to a series of remedial actions.
* In September 2012 the CEO of a major smartphone manufacturer released a letter apologizing for the poor quality of a new widely-used mapping application.
* Problems with new trading software installed by a major equities market-maker resulted in a one-day loss to the company of more than $400 million according to news reports in August of 2012. Stock market activity in many stocks was significantly disrupted. Five months after the event, the market-making company's own stock price was still down more than 60%.
* In July 2012 an entire 17-minute 7000-shell fireworks show was unintentionally set off all at once at the start of the display, reportedly due to a glitch in the computer system controlling the fireworks sequencing.
* A bug in a major operating system's handling of 'leap seconds' (an occasional adjustment to the world's atomic clocks) resulted in system problems reported worldwide in July of 2012. Although a fix for the bug had been developed earlier in the year, some versions of the OS had not yet been patched.
* A software failure at a large European bank resulted in millions of customers being unable to access their money for four days in June 2012, according to media reports. The problem occurred after a software upgrade and was due to either poor testing or poor contingency planning, according to the reports.
* In March of 2012 the Initial Public Offering of the stock of a new stock exchange was cancelled due to software bugs in their trading platform that interfered with trading in stocks including their own IPO stock, according to media reports. The high-speed trading platform reportedly was already handling more than 10 percent of all trading in U.S. securities, but the processing of initial IPO trading was new for the system, and though it had undergone testing, it was unable to properly handle the IPO initial trades. The problem also briefly affected trading of other stocks and other stock exchanges.
* A leap day bug was reported to have caused interruption of service to many customers of a major public cloud infrastructure provider in February 2012. The company subsequently stated that they would be taking steps to improve their testing.
* It was reported that software problems in an automated highway toll charging system caused erroneous charges to thousands of customers in a short period of time in December 2011.
* A U.S. county found that their state's computer software assigned thousands of voters to invalid voting locations in November 2011 for an upcoming election due to the system's problems accepting new voting district boundary information.
* In August 2011, a major North American retailer initiated its own online e-commerce website, after contracting it out for many years. It was reported that within the first few months the site crashed six times, home page links were found not to work, gift registries were reported not working properly, and the online division's president left the company.
* A new U.S.-government-run credit card complaint handling system was not working correctly according to August 2011 news reports. Banks were required to respond to complaints routed to them from the system, but due to system bugs the complaints were not consistently being routed to companies as expected. Reportedly the system had not been properly tested.
* News reports in Asia in July of 2011 reported that software bugs in a national computerized testing and grading system resulted in incorrect test results for tens of thousands of high school students. The national education ministry had to reissue grade reports to nearly 2 million students nationwide.
* In mid-2011 it was reported that expensive new provincial government court system software had thousands of bugs during its first year of operation that caused errors such as incorrect dates for suspension of drivers licenses, adults being sentenced in juvenile courts, incorrect records as to whether a defendant had shown up in court, and incorrect information in warrants.
* In April of 2011 bugs were found in popular smartphone software that resulted in long-term data storage on the phone that could be utilized in location tracking of the phone, even when it was believed that locator services in the phone were turned off. A software update was released several weeks later which was expected to resolve the issues.
* In March 2011 a major Asian bank experienced computer system failures resulting in thousands of ATM's being unavailable, internet banking unavailable for 3 days, delays in salary payments to hundreds of thousands of workers, and more than $10 billion in failed transactions, according to new reports. The cause was attributed to the system's inability to handle a surge in transactions. The bank had to consult with rival banks for help in dealing with the huge numnber of failed transactions, and within a few months the bank's president and head of IT both resigned.
* A securities regulatory agency required an investment company to pay a $25 million fine "...for concealing a significant error in the computer code..." and to repay clients $217 million "...to redress harm from the coding error..." according to the regulatory agency's web site in February 2011. The coding errors were stated to be in the quantitative investment model used by the investment company to manage client investments.
* Software problems in a new software upgrade for farecards in a major urban transit system reportedly resulted in a loss of a half million dollars before the software was fixed, according to October 2010 news reports.
* In October of 2010 a large municipality's new web-based election voting system was opened to the public for a testing period in which users were invited to attempt to break it. Within a few days the site was penetrated by college student hackers and its functionality altered.
* A game software company released a new product in mid-2010 that was reportedly so buggy that the CEO sent customers a letter apologizing for the initial poor quality of the game.
* A smartphone online banking application was reported in July 2010 to have a security bug affecting more than 100,000 customers. Users were able to upgrade to a newer software version that fixed the problem.
* In July 2010 a major smartphone maker reported that their software contained a long-time bug that resulted in incorrect indicators of signal strength in the phone's interface. Reportedly customers had been complaining about the problem for several years. The company provided a fix for the problem several weeks later.
* News reports in April 2010 indicated that a major antivirus software vendor provided a faulty signature update file which caused computers to crash, continuously reboot, or lose network connectivity. This was reportedly due to a problematical change in the vendor's testing process. Stories of affected systems included police departments reduced to hand-written reports, hospitals turning away patients, and closing of supermarkets. The software vendor was sold within a year and was no longer an independent company.
* A major auto manufacturer was reported to have found that a software problem was the cause of vehicle braking delayed reactions in one of its popular models, according to February 2010 media reports.
* Email services of a major smartphone system were interrupted or unavailable for nine hours in December 2009, the second service interruption within a week, according to news reports. The problems were believed to be due to bugs in new versions of the email system software.
* It was reported in August 2009 that a large suburban school district introduced a new computer system that was 'plagued with bugs' and resulted in many students starting the school year without schedules or with incorrect schedules, and many problems with grades. Upset students and parents started a social networking site for sharing complaints.
* In February of 2009 users of a major search engine site were prevented from clicking through to sites listed in search results for part of a day. It was reportedly due to software that did not effectively handle a mistakenly-placed "/" in an internal ancillary reference file that was frequently updated for use by the search engine. Users, instead of being able to click thru to listed sites, were instead redirected to an intermediary site which, as a result of the suddenly enormous load, was rendered unusable.
* A large health insurance company was reportedly banned by regulators from selling certain types of insurance policies in January of 2009 due to ongoing computer system problems that resulted in denial of coverage for needed medications and mistaken overcharging or cancelation of benefits. The regulatory agency was quoted as stating that the problems were posing "a serious threat to the health and safety" of beneficiaries.
* A news report in January 2009 indicated that a major IT and management consulting company was still battling years of problems in implementing its own internal accounting systems, including a 2005 implementation that reportedly "was attempted without adequate testing".
* In August of 2008 it was reported that more than 600 U.S. airline flights were significantly delayed due to a software glitch in the U.S. FAA air traffic control system. The problem was claimed to be a 'packet switch' that 'failed due to a database mismatch', and occurred in the part of the system that handles required flight plans.
* Software system problems at a large health insurance company in August 2008 were the cause of a privacy breach of personal health information for several hundred thousand customers, according to news reports. It was claimed that the problem was due to software that 'was not comprehensively tested'.
* A major clothing retailer was reportedly hit with significant software and system problems when attempting to upgrade their online retailing systems in June 2008. Problems remained ongoing for some time. When the company made their public quarterly financial report, the software and system problems were claimed as the cause of the poor financial results.
* Software problems in the automated baggage sorting system of a major airport in February 2008 prevented thousands of passengers from checking baggage for their flights. It was reported that the breakdown occurred during a software upgrade, despite pre-testing of the software. The system continued to have problems in subsequent months.
* News reports in December of 2007 indicated that significant software problems were continuing to occur in a new ERP payroll system for a large urban school system. It was believed that more than one third of employees had received incorrect paychecks at various times since the new system went live the preceding January, resulting in overpayments of $53 million, as well as underpayments. An employees' union brought a lawsuit against the school system, the cost of the ERP system was expected to rise by 40%, and the non-payroll part of the ERP system was delayed. Inadequate testing reportedly contributed to the problems. The school system was still working on cleaning up the aftermath of the problems in December 2009, going so far as to bring lawsuits against some employees to get them to return overpayments.
* In November of 2007 a regional government reportedly brought a multi-million dollar lawsuit against a software services vendor, claiming that the vendor 'minimized quality' in delivering software for a large criminal justice information system and the system did not meet requirements. The vendor also sued its subcontractor on the project.
* In June of 2007 news reports claimed that software flaws in a popular online stock-picking contest could be used to gain an unfair advantage in pursuit of the game's large cash prizes. Outside investigators were called in and in July the contest winner was announced. Reportedly the winner had previously been in 6th place, indicating that the top 5 contestants may have been disqualified.
* A software problem contributed to a rail car fire in a major underground metro system in April of 2007 according to newspaper accounts. The software reportedly failed to perform as expected in detecting and preventing excess power usage in equipment on new passenger rail cars, resulting in overheating and fire in the rail car, and evacuation and shutdown of part of the system.
* Tens of thousands of medical devices were recalled in March of 2007 to correct a software bug. According to news reports, the software would not reliably indicate when available power to the device was too low.
* A September 2006 news report indicated problems with software utilized in a state government's primary election, resulting in periodic unexpected rebooting of voter checkin machines, which were separate from the electronic voting machines, and resulted in confusion and delays at voting sites. The problem was reportedly due to insufficient testing.
* In August of 2006 a U.S. government student loan service erroneously made public the personal data of as many as 21,000 borrowers on it's web site, due to a software error. The bug was fixed and the government department subsequently offered to arrange for free credit monitoring services for those affected.
* A software error reportedly resulted in overbilling of up to several thousand dollars to each of 11,000 customers of a major telecommunications company in June of 2006. It was reported that the software bug was fixed within days, but that correcting the billing errors would take much longer.
* News reports in May of 2006 described a multi-million dollar lawsuit settlement paid by a healthcare software vendor to one of its customers. It was reported that the customer claimed there were problems with the software they had contracted for, including poor integration of software modules, and problems that resulted in missing or incorrect data used by medical personnel.
* In early 2006 problems in a government's financial monitoring software resulted in incorrect election candidate financial reports being made available to the public. The government's election finance reporting web site had to be shut down until the software was repaired.
* Trading on a major Asian stock exchange was brought to a halt in November of 2005, reportedly due to an error in a system software upgrade. The problem was rectified and trading resumed later the same day.
* A May 2005 newspaper article reported that a major hybrid car manufacturer had to install a software fix on 20,000 vehicles due to problems with invalid engine warning lights and occasional stalling. In the article, an automotive software specialist indicated that the automobile industry spends $2 billion to $3 billion per year fixing software problems.
* Media reports in January of 2005 detailed severe problems with a $170 million high-profile U.S. government IT systems project. Software testing was one of the five major problem areas according to a report of the commission reviewing the project. In March of 2005 it was decided to scrap the entire project.
* In July 2004 newspapers reported that a new government welfare management system in Canada costing several hundred million dollars was unable to handle a simple benefits rate increase after being put into live operation. Reportedly the original contract allowed for only 6 weeks of acceptance testing and the system was never tested for its ability to handle a rate increase.
* Millions of bank accounts were impacted by errors due to installation of inadequately tested software code in the transaction processing system of a major North American bank, according to mid-2004 news reports. Articles about the incident stated that it took two weeks to fix all the resulting errors, that additional problems resulted when the incident drew a large number of e-mail phishing attacks against the bank's customers, and that the total cost of the incident could exceed $100 million.
* A bug in site management software utilized by companies with a significant percentage of worldwide web traffic was reported in May of 2004. The bug resulted in performance problems for many of the sites simultaneously and required disabling of the software until the bug was fixed.
* According to news reports in April of 2004, a software bug was determined to be a major contributor to the 2003 Northeast blackout, the worst power system failure in North American history. The failure involved loss of electrical power to 50 million customers, forced shutdown of 100 power plants, and economic losses estimated at $6 billion. The bug was reportedly in one utility company's vendor-supplied power monitoring and management system, which was unable to correctly handle and report on an unusual confluence of initially localized events. The error was found and corrected after examining millions of lines of code.
* In early 2004, news reports revealed the intentional use of a software bug as a counter-espionage tool. According to the report, in the early 1980's one nation surreptitiously allowed a hostile nation's espionage service to steal a version of sophisticated industrial software that had intentionally-added flaws. This eventually resulted in major industrial disruption in the country that used the stolen flawed software.
* A major U.S. retailer was reportedly hit with a large government fine in October of 2003 due to web site errors that enabled customers to view one another's online orders.
* News stories in the fall of 2003 stated that a manufacturing company recalled all their transportation products in order to fix a software problem causing instability in certain circumstances. The company found and reported the bug itself and initiated the recall procedure in which a software upgrade fixed the problems.
* In August of 2003 a U.S. court ruled that a lawsuit against a large online brokerage company could proceed; the lawsuit reportedly involved claims that the company was not fixing system problems that sometimes resulted in failed stock trades, based on the experiences of 4 plaintiffs during an 8-month period. A previous lower court's ruling that "...six miscues out of more than 400 trades does not indicate negligence." was invalidated.
* In April of 2003 it was announced that a large student loan company in the U.S. made a software error in calculating the monthly payments on 800,000 loans. Although borrowers were to be notified of an increase in their required payments, the company will still reportedly lose $8 million in interest. The error was uncovered when borrowers began reporting inconsistencies in their bills.
* News reports in February of 2003 revealed that the U.S. Treasury Department mailed 50,000 Social Security checks without any beneficiary names. A spokesperson indicated that the missing names were due to an error in a software change. Replacement checks were subsequently mailed out with the problem corrected, and recipients were then able to cash their Social Security checks.
* It was reported that in April 2002, problems with the integration of several merged bank systems in Japan resulted in millions of errors in ATM transactions, automatic bill payments errors, delayed debits, duplicate debits, and other problems. Reportedly the problems were caused by a delay in the start of the systems integration work and subsequent inadequate testing, and it took more than a month to restore banking operations to normal
* In March of 2002 it was reported that software bugs in Britain's national tax system resulted in more than 100,000 erroneous tax overcharges. The problem was partly attributed to the difficulty of testing the integration of multiple systems.
* A newspaper columnist reported in July 2001 that a serious flaw was found in off-the-shelf software that had long been used in systems for tracking certain U.S. nuclear materials. The same software had been recently donated to another country to be used in tracking their own nuclear materials, and it was not until scientists in that country discovered the problem, and shared the information, that U.S. officials became aware of the problems.
* According to newspaper stories in mid-2001, a major systems development contractor was fired and sued over problems with a large retirement plan management system. According to the reports, the client claimed that system deliveries were late, the software had excessive defects, and it caused other systems to crash.
* In January of 2001 newspapers reported that a major European railroad was hit by the aftereffects of the Y2K bug. The company found that many of their newer trains would not run due to their inability to recognize the date '31/12/2000'; the trains were started by altering the control system's date settings.
* News reports in September of 2000 told of a software vendor settling a lawsuit with a large mortgage lender; the vendor had reportedly delivered an online mortgage processing system that did not meet specifications, was delivered late, and didn't work.
* In early 2000, major problems were reported with a new computer system in a large suburban U.S. public school district with 100,000+ students; problems included 10,000 erroneous report cards and students left stranded by failed class registration systems; the district's CIO was fired. The school district decided to reinstate its original 25-year old system for at least a year until the bugs were worked out of the new system by the software vendors.
* A review board concluded that the NASA Mars Polar Lander failed in December 1999 due to software problems that caused improper functioning of retro rockets utilized by the Lander as it entered the Martian atmosphere.
* During an attempt to put a commercial sateliite into orbit in October 1999, the 2nd launch of a new private rocket launch business reportedly failed due to a software error that caused problems in a valve in the rocket's second-stage.
* In October of 1999 the $125 million NASA Mars Climate Orbiter spacecraft was believed to be lost in space due to a simple data conversion error. It was determined that spacecraft software used certain data in English units that should have been in metric units. Among other tasks, the orbiter was to serve as a communications relay for the Mars Polar Lander mission, which failed for unknown reasons in December 1999. Several investigating panels were convened to determine the process failures that allowed the error to go undetected.
* Bugs in software supporting a large commercial high-speed data network affected 70,000 business customers over a period of 8 days in August of 1999. Among those affected was the electronic trading system of the largest U.S. futures exchange, which was shut down for most of a week as a result of the outages.
* In April of 1999 a software bug caused the failure of a $1.2 billion U.S. military satellite launch, the costliest unmanned accident in the history of Cape Canaveral launches. The failure was the latest in a string of launch failures, triggering a complete military and industry review of U.S. space launch programs, including software integration and testing processes. Congressional oversight hearings were requested.
* A small town in Illinois in the U.S. received an unusually large monthly electric bill of $7 million in March of 1999. This was about 700 times larger than its normal bill. It turned out to be due to bugs in new software that had been purchased by the local power company to deal with Y2K software issues.
* In early 1999 a major computer game company recalled all copies of a popular new product due to software problems. The company made a public apology for releasing a product before it was ready.
* The computer system of a major online U.S. stock trading service failed during trading hours several times over a period of days in February of 1999 according to nationwide news reports. The problem was reportedly due to bugs in a software upgrade intended to speed online trade confirmations.
* In April of 1998 a major U.S. data communications network failed for 24 hours, crippling a large part of some U.S. credit card transaction authorization systems as well as other large U.S. bank, retail, and government data systems. The cause was eventually traced to a software bug.
* January 1998 news reports told of software problems at a major U.S. telecommunications company that resulted in no charges for long distance calls for a month for 400,000 customers. The problem went undetected until customers called up with questions about their bills.
* In November of 1997 the stock of a major health industry company dropped 60% due to reports of failures in computer billing systems, problems with a large database conversion, and inadequate software testing. It was reported that more than $100,000,000 in receivables had to be written off and that multi-million dollar fines were levied on the company by government agencies.
* A retail store chain filed suit in August of 1997 against a transaction processing system vendor (not a credit card company) due to the software's inability to handle credit cards with year 2000 expiration dates.
* In August of 1997 one of the leading consumer credit reporting companies reportedly shut down their new public web site after less than two days of operation due to software problems. The new site allowed web site visitors instant access, for a small fee, to their personal credit reports. However, a number of initial users ended up viewing each others' reports instead of their own, resulting in irate customers and nationwide publicity. The problem was attributed to "...unexpectedly high demand from consumers and faulty software that routed the files to the wrong computers."
* In November of 1996, newspapers reported that software bugs caused the 411 telephone information system of one of the U.S. RBOC's to fail for most of a day. Most of the 2000 operators had to search through phone books instead of using their 13,000,000-listing database. The bugs were introduced by new software modifications and the problem software had been installed on both the production and backup systems. A spokesman for the software vendor reportedly stated that 'It had nothing to do with the integrity of the software. It was human error.'
* On June 4 1996 the first flight of the European Space Agency's new Ariane 5 rocket failed shortly after launching, resulting in an estimated uninsured loss of a half billion dollars. It was reportedly due to the lack of exception handling of a floating-point error in a conversion from a 64-bit integer to a 16-bit signed integer.
* Software bugs caused the bank accounts of 823 customers of a major U.S. bank to be credited with $924,844,208.32 each in May of 1996, according to newspaper reports. The American Bankers Association claimed it was the largest such error in banking history. A bank spokesman said the programming errors were corrected and all funds were recovered.
* In August 1991 the concrete base structure for a North Sea oil platform imploded and sank off the coast of Norway, reportedly due to errors in initially-used design software. The enormous structure, on hitting the seabed, reportedly was detected as a magnitude 3.0 seismic event and resulted in a loss of $700 million. The base structure was eventually redesigned and the full platform was completed two years later, and was still in use as of 2008.
* On January 1 1984 all computers produced by one of the leading minicomputer makers of the time reportedly failed worldwide. The cause was claimed to be a leap year bug in a date handling function utilized in deletion of temporary operating system files. Technicians throughout the world worked for several days to clear up the problem. It was also reported that the same bug affected many of the same computers four years later.
* Software bugs in a Soviet early-warning monitoring system nearly brought on nuclear war in 1983, according to news reports in early 1999. The software was supposed to filter out false missile detections caused by Soviet satellites picking up sunlight reflections off cloud-tops, but failed to do so. Disaster was averted when a Soviet commander, based on what he said was a '...funny feeling in my gut', decided the apparent missile attack was a false alarm. The filtering software code was rewritten. The Soviet commander, Stanislav Petrov, passed away at home in his apartment in a Moscow suburb at age 77 on May 19 2017.

For more lists of software bugs see ['Collection of Software Bugs'](http://www5.in.tum.de/~huckle/bugse.html#WWW), a large collection of bugs and links to other bug lists maintained by Prof. Thomas Huckle at the Institut für Informatik in Germany, and a ['List of software bugs'](http://en.wikipedia.org/wiki/List_of_notable_software_bugs) in various categories maintained on Wikipedia.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**Does every software project need testers?**  
While all projects will benefit from testing, some projects may not require independent test staff to succeed.

Which projects may not need independent test staff? The answer depends on the size and context of the project, the risks, the development methodology, the skill and experience of the developers, and other factors. For instance, if the project is a short-term, small, low risk project, with highly experienced programmers utilizing thorough unit testing or test-first development, then test engineers may not be required for the project to succeed.

In some cases an IT organization may be too small or new to have a testing staff even if the situation calls for it. In these circumstances it may be appropriate to instead use contractors or outsourcing, or adjust the project management and development approach (by switching to more senior developers and test-first development, for example). Inexperienced managers sometimes gamble on the success of a project by skipping thorough testing or having programmers do post-development functional testing of their own work, a decidedly high risk gamble.

For non-trivial-size projects or projects with non-trivial risks, a testing staff is usually necessary. As in any business, the use of personnel with specialized skills enhances an organization's ability to be successful in large, complex, or difficult tasks. It allows for both a) deeper and stronger skills and b) the contribution of differing perspectives. For example, programmers typically have the perspective of 'what are the technical issues in making this functionality work?'. A test engineer typically has the perspective of 'what might go wrong with this functionality, and how can we ensure it meets expectations?'. A technical person who can be highly effective in approaching tasks from both of those perspectives is rare, which is why, sooner or later, organizations bring in test specialists.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**Why does software have bugs?**

* miscommunication or no communication - as to specifics of what an application should or shouldn't do (the application's requirements).
* software complexity - the complexity of current software applications can be difficult to comprehend for anyone without experience in modern-day software development. Multi-tier distributed systems, applications utilizing multiple local and remote web services, use of cloud infrastructure, data communications, enormous/distributed datastores, security complexities, and sheer size of applications have all contributed to the exponential growth in software/system complexity.
* programming errors - programmers, like anyone else, can make mistakes.
* dependencies among code modules, services, systems, other projects, etc may not be well understood, and may cause unexpected problems.
* in some fast-changing business environments, continuously changing specifications may be a fact of life, thus introducing significant added risk. Agile software development approaches - if effectively implemented - can help mitigate this. See more about ['agile' approaches](http://www.softwareqatest.com/qatfaq2.html#FAQ2_17)in Part 2 of the FAQ.
* time pressures - scheduling of software projects is difficult at best, often requiring a lot of guesswork. When deadlines loom and the crunch comes, mistakes will be made.
* egos - people prefer to say things like:
* 'no problem'
* 'piece of cake'
* 'I can whip that out in a few hours'
* 'it should be easy to update that old code'
* instead of:
* 'that adds a lot of complexity and we could end up
* making a lot of mistakes'
* 'we have no idea if we can do that; we'll wing it'
* 'I can't estimate how long it will take, until I
* take a close look at it'
* 'we can't figure out what that old spaghetti code
* did in the first place'
* If there are too many unrealistic 'no problem's', the
* result may be bugs.
* poorly designed/documented code - it's tough to maintain and modify code that is badly written or poorly commented/documented; the result is bugs. In many organizations management provides no incentive for programmers to write clear, understandable, maintainable code. In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it ('if it was hard to write, it should be hard to read').
* software development tools - IDE's, libraries, external apps/services, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, or have usability issues, resulting in added bugs.
* services or microservices on which the software depends also often introduce their own bugs or performance problems, are not well understood, or may be unreliable, resulting in added bugs.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**How can new Software QA processes be introduced in an existing organization?**

* A lot depends on the size of the organization and the risks involved. For large organizations with high-risk (in terms of lives or property) projects, serious management buy-in is required and a more formalized QA process may be necessary.
* Where the risk is lower, management and organizational buy-in and QA implementation may be a slower, step-at-a-time process. QA processes should be balanced with productivity so as to keep bureaucracy from getting out of hand.
* For small groups or projects, a more ad-hoc process may be appropriate, depending on the type of customers and projects. A lot will depend on team leads or managers, feedback to/from developers, and ensuring adequate communications among customers, managers, developers, testers, and other stakeholders.
* The most value for effort will often be in (a) requirement/user story management processes, with a goal of clear, complete, testable specifications embodied in requirements, appropriately-sized user stories, or design documentation, (b) design reviews and code reviews, and (c) post-mortems/retrospectives. Agile approaches utilizing extensive regular communication among the development team and product owner and other stakeholders can coordinate well with improved QA processes.
* Other possibilities include incremental approaches such as Lean/Kaizen methods of continuous process improvement, the Deming-Shewhart Plan-Do-Check-Act cycle, and others.

Also see ['How can QA processes be implemented without reducing productivity?'](http://www.softwareqatest.com/qat_lfaq1.html#LFAQ1_6) in the LFAQ section.

(See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA', 'Software Engineering', and 'Project Management' categories for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is verification? validation?**  
Verification typically involves reviews and meetings to evaluate documents, plans, code, requirements, and specifications. This can be done with checklists, issues lists, walkthroughs, and inspection meetings. Validation typically involves actual testing and takes place after verifications are completed. The term 'IV & V' refers to Independent Verification and Validation.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is a 'walkthrough'?**  
A 'walkthrough' is an informal meeting for evaluation or informational purposes. Little or no preparation is usually required.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What's an 'inspection'?**  
An inspection is more formalized than a 'walkthrough', typically with 3-8 people including a moderator, reader, and a recorder to take notes. The subject of the inspection is typically a document such as a requirements spec or a test plan, and the purpose is to find problems and see what's missing, not to fix anything. Attendees should prepare for this type of meeting by reading thru the document; most problems will be found during this preparation. The result of the inspection meeting should be a written report. Thorough preparation for inspections is difficult, painstaking work, but is one of the most cost effective methods of ensuring quality. Employees who are most skilled at inspections are like the 'eldest brother' in the parable in ['Why is it often hard for organizations to get serious about quality assurance?'](http://www.softwareqatest.com/qat_lfaq1.html#LFAQ1_1). Their skill may have low visibility but they are extremely valuable to any software development organization, since bug prevention is far more cost-effective than bug detection.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What kinds of testing should be considered?**

* black box testing - not based on any knowledge of internal design or code. Tests are based on requirements and functionality.
* white box testing - based on knowledge of the internal logic of an application's code. Tests are based on coverage of code statements, branches, paths, conditions.
* unit testing - the most 'micro' scale of testing; to test particular functions or code modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. Not always easily done unless the application has a well-designed architecture with tight code; may require developing test driver modules or test harnesses.
* API testing - testing of messaging/data exchange among systems or components of systems. Such testing usually does not involve GUI's (graphical user interfaces). It is often considered a type of 'mid-level' testing.
* incremental integration testing - continuous testing of an application as new functionality is added; requires that various aspects of an application's functionality be independent enough to work separately before all parts of the program are completed, or that test drivers be developed as needed; done by programmers or by testers.
* integration testing - testing of combined parts of an application to determine if they function together correctly. The 'parts' can be code modules, services, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to multi-tier and distributed systems.
* functional testing - black-box type testing geared to functional requirements of an application; this type of testing should be done by testers. This doesn't mean that the programmers shouldn't check that their code works before releasing it (which of course applies to any stage of testing.)
* system testing - black-box type testing that is based on overall requirements specifications; covers all combined parts of a system.
* end-to-end testing - similar to system testing; the 'macro' end of the test scale; involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.
* sanity testing or smoke testing - typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing systems every 5 minutes, bogging down systems to a crawl, or corrupting databases, the software may not be in a 'sane' enough condition to warrant further testing in its current state.
* regression testing - re-testing after fixes or modifications of the software or its environment. It can be difficult to determine how much re-testing is needed, especially near the end of the development cycle. Automated testing approaches can be especially useful for this type of testing.
* acceptance testing - final testing based on specifications of the end-user or customer, or based on use by end-users/customers over some limited period of time.
* load testing - testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.
* stress testing - term often used interchangeably with 'load' and 'performance' testing. Also used to describe such tests as system functional testing while under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database system, etc.
* performance testing - term often used interchangeably with 'stress' and 'load' testing. Ideally 'performance' testing (and any other 'type' of testing) is defined in requirements documentation or QA or Test Plans.
* usability testing - testing for 'user-friendliness'. Clearly this is subjective, and will depend on the targeted end-user or customer. User interviews, surveys, video recording of user sessions, and other techniques can be used. Programmers and testers are usually not appropriate as usability testers.
* accessibility testing (sometimes called '508 testing', in reference to Section 508 of a U.S. federal law, covering government-related software systems), is a type of usability testing oriented toward users with disabilites.
* install/uninstall testing - testing of full, partial, or upgrade install/uninstall processes.
* recovery testing - testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.
* failover testing - typically used interchangeably with 'recovery testing'
* security testing - testing how well the system protects against unauthorized internal or external access, willful damage, etc; may require sophisticated testing techniques.
* compatibility testing - testing how well software performs in a particular hardware/software/operating system/network/etc. environment.
* exploratory testing - often taken to mean a creative, informal software test that is not based on formal test plans or test cases; testers may be learning the software as they test it.
* ad-hoc testing - similar to exploratory testing, but often taken to mean that the testers have significant understanding of the software before testing it.
* context-driven testing - testing driven by an understanding of the environment, culture, and intended use of software. For example, the testing approach for life-critical medical equipment software would be completely different than that for a low-cost computer game.
* user acceptance testing - determining if software is satisfactory to an end-user or customer.
* comparison testing - comparing software weaknesses and strengths to competing products.
* alpha testing - testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Typically done by end-users or others, not by programmers or testers.
* beta testing - testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Typically done by end-users or others, not by programmers or testers.
* mutation testing - a method for determining if a set of test data or test cases is useful, by deliberately introducing various code changes ('bugs') and retesting with the original test data/cases to determine if the 'bugs' are detected. Proper implementation requires large computational resources.

(See the [Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software Testing' category for useful books on Software Testing.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What are 5 common problems in the software development process?**

* poor requirements or user stories - if these are unclear, incomplete, too general, or not testable, there may be problems.
* unrealistic schedule or story points - if too much work is crammed in too little time, problems are inevitable.
* inadequate testing - no one may know whether or not the software is any good until customers complain or systems crash.
* misunderstandings about dependencies.
* miscommunication - if developers don't know what's needed or stakeholders have erroneous expectations, problems can be expected.

In agile projects, problems often occur when the project diverges from agile principles (such as forgetting that 'Business people and developers must work together daily throughout the project.' or 'The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.' - see the[Manifesto for Agile Software Development](http://agilemanifesto.org/principles.html).)

(See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA', 'Software Engineering', and 'Project Management' categories for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What are 5 common solutions to software development problems?**

* solid requirements/user stories - clear, complete, appropriately detailed, cohesive, attainable, testable specifications or acceptance criteria that are agreed to by all players. In 'agile'-type environments, continuous close coordination with product owners or their representatives is necessary to ensure that changing/emerging requirements are understood.
* realistic schedules - allow adequate time for planning, design, testing, bug fixing, re-testing, changes, and documentation; personnel should be able to complete the project without burning out, and be able to work at a sustainable pace.
* adequate testing - start testing early on, re-test after fixes or changes, plan for adequate time for testing and bug-fixing. 'Early' testing could include static code analysis/testing, test-first development, unit testing by developers, built-in testing and diagnostic capabilities, etc. Automated testing can contribute significantly if effectively designed and implemented as part of an overall testing strategy.
* stick to initial requirements/criteria where feasible - be prepared to defend against excessive changes and additions once development has begun or after a sprint has begun, and be prepared to explain consequences. If changes are necessary, they should be adequately reflected in related schedule changes or story/point changes. If possible, work closely with customers/end-users to manage expectations. In agile environments, it is acceptable that requirements may change often, requiring that true agile processes be in place and followed. Note that in true agile practices stories should not change during a sprint.
* communication - require walkthroughs/inspections/reviews when appropriate; make extensive use of group communication tools - groupware, wiki's, bug-tracking tools, change management tools, audio/video conferencing, etc.; ensure that information/documentation/user stories are available, up-to-date, and appopriately detailed; promote teamwork and cooperation; use prototypes, frequent deliveries, and/or continuous communication with end-users if possible to clarify expectations. In effective agile environments most of these should be taking place.

(See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA', 'Software Engineering', and 'Project Management' categories for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is software 'quality'?**  
Quality software is reasonably bug-free, delivered on time and within budget, meets requirements, acceptance criteria, and/or expectations, and is maintainable. However, quality is obviously a subjective term. It will depend on who the 'customer' is and their overall influence in the scheme of things. A wide-angle view of the 'customers' of a software development project might include end-users, product owners, customer acceptance testers, customer contract officers, customer management, the development organization's management/accountants/testers/salespeople, future software maintenance engineers, stockholders, magazine columnists, etc. Each type of 'customer' will have their own slant on 'quality' - the accounting department might define quality in terms of profits while an end-user might define quality as user-friendly and bug-free. (See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA' category for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is 'good code'?**  
'Good code' is code that works, is reasonably bug free, secure, and is readable and maintainable. Some organizations have coding 'standards' that all developers are supposed to adhere to, but everyone has different ideas about what's best, or what is too many or too few rules. There are also various theories and metrics, such as McCabe Complexity metrics. It should be kept in mind that excessive use of standards and rules can stifle productivity and creativity. 'Peer reviews', 'buddy checks' pair programming, code analysis tools, etc. can be used to check for problems and enforce standards.   
For example, in C/C++ coding, here are some typical ideas to consider in setting rules/standards; these may or may not apply to a particular situation:

* minimize or eliminate use of global variables.
* use descriptive function and method names - use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions.
* use descriptive variable names - use both upper and lower case, avoid abbreviations, use as many characters as necessary to be adequately descriptive (use of more than 20 characters is not out of line); be consistent in naming conventions.
* function and method sizes should be minimized; less than 100 lines of code is good, less than 50 lines is preferable.
* function/method descriptions should be clearly spelled out in comments preceding a function's/method's code.
* organize code for readability.
* use whitespace generously - vertically and horizontally
* each line of code should contain 70 characters max.
* one code statement per line.
* coding style should be consistent throughout a program (e.g., use of brackets, indentations, naming conventions, etc.)
* in adding comments, err on the side of too many rather than too few comments; a common rule of thumb is that there should be at least as many lines of comments (including header blocks) as lines of code.
* no matter how small, an application should include documentation of the overall program function and flow (even a few paragraphs is better than nothing); or if possible a separate flow chart and detailed program documentation.
* make extensive use of error handling procedures and status and error logging.
* for C++, to minimize complexity and increase maintainability, avoid too many levels of inheritance in class hierarchies (relative to the size and complexity of the application). Minimize use of multiple inheritance, and minimize use of operator overloading (note that the Java programming language eliminates multiple inheritance and operator overloading.)
* for C++, keep class methods small, less than 50 lines of code per method is preferable.
* for C++, make liberal use of exception handlers

Also see [Google's collection of code style guides for many different languages](https://github.com/google/styleguide), which can be useful in considering your particular code guidelines/styles.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is 'good design'?**  
'Design' could refer to many things, but often refers to 'functional design' or 'internal design'. Good internal design is indicated by software code whose overall structure is clear, understandable, easily modifiable, and maintainable; is robust with sufficient error-handling and status logging capability; and works as expected when implemented. Good functional design is indicated by an application whose functionality can be traced back to customer and end-user requirements or user stories. (See further discussion of functional and internal design in FAQ ['What's the big deal about requirements?'](http://www.softwareqatest.com/qatfaq2.html#FAQ2_5)). For programs that have a user interface, it's often a good idea to assume that the end user will have little computer knowledge and may not read a user manual or even the on-line help; some common rules-of-thumb include:

* the program should act in a way that least surprises the user
* it should always be evident to the user what can be done next and how to exit
* the program shouldn't let the users do something stupid without warning them.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is SEI? CMM? CMMI? ISO? IEEE? ANSI? Will it help?**

* SEI = 'Software Engineering Institute' at Carnegie-Mellon University; initiated by the U.S. Defense Department to help improve software development processes.
* CMM = 'Capability Maturity Model', now called the CMMI ('Capability Maturity Model Integration'), developed by the SEI and as of January 2013 overseen by the CMMI Institute at Carnegie Mellon University. In the 'staged' version, it's a model of 5 levels of process 'maturity' that help determine effectiveness in delivering quality software. CMMI models are "collections of best practices that help organizations to improve their processes." It is geared to larger organizations such as large U.S. Defense Department contractors. However, many of the QA processes involved are appropriate to any organization, and if reasonably applied can be helpful. Organizations can receive CMMI ratings by undergoing assessments by qualified auditors. CMMI V1.3 (2010) also supports Agile development processes. See the searchable [CMMI assessment results database](https://sas.cmmiinstitute.com/pars/pars.aspx).

Level 1 - 'Initial': characterized by chaos, periodic panics, and heroic efforts

required by individuals to successfully complete projects. Few if any

processes in place; successes may not be repeatable.

Level 2 - 'Managed': projects carried out in accordance with policies and employ

skilled personnel with sufficient resources. Project tracking and reporting

is in place. Schedules and budgets are set and revised as needed. Work

products are appropriately controlled.

Level 3 - 'Defined': standard development and maintenance processes are established,

integrated consistently throughout an organization,

Level 4 - 'Quantitatively Managed': metrics are used to track process performance.

Project performance is controlled and predictable.

Level 5 - 'Optimizing': the focus is on continuous process improvement. The impact of

new processes and technologies can be predicted and effectively implemented

when required. Quality and process objectives are established and regularly

revised to reflect changing objectives and organizational performance, and

used as criteria in managing process improvement.

Perspective on CMMI ratings: During 2017, of 2800 organizations appraised,

it was reported that 10% of appraisals were at Level 4 or 5; 70% of appraised

organizations had less than 100 employees; 50% of appraisals were in China,

other Asia 20%, and 20% in N. America; and 80% were using agile methodologies.

During 2002-2005, of 782 organizations assessed, 4% were rated at Level 1,

34% at 2, 30% at 3, 4% at 4, and 19% at Level 5. 37% were government

contractors or agencies, and 68% had more than 100 employees; 88% were software-related

organizations. (The CMMI applies to a wide variety of organizations, not just software

organizations.) The majority of organizations were NOT U.S.-based.

* ISO = 'International Organisation for Standardization' - The ISO 9001:2015 standard (the ISO standard is updated periodically, indicated by the year designation ) concerns quality systems that are assessed by outside auditors, and it applies to many kinds of production and manufacturing organizations, not just software. It covers documentation, design, development, production, testing, installation, servicing, and other processes. The full set of standards consists of: (a)ISO 9001:2015 - Quality Management Systems: Requirements; (b)ISO 9000:2015 - Quality Management Systems: Fundamentals and Vocabulary; (c)ISO 9004:2009 - Quality Management Systems: Guidelines for Performance Improvements. (d)ISO 19011:2011 - Guidelines for auditing management systems. To be ISO 9001 certified, a third-party auditor assesses an organization, and certification is typically good for about 3 years, after which a complete reassessment is required. Note that ISO certification does not necessarily indicate quality products - it indicates only that documented processes are followed. There are also other software-related ISO standards such as ISO/IEC 25010 which includes a 'quality in use model' composed of five characteristics and a 'product quality model' that covers eight main characteristics of software. Also see <http://www.iso.org/> for the latest information. In the U.S. the standards can also be purchased via the ASQ web site at <http://asq.org/quality-press/>  
  [ISO/IEC 25010](http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733) is a software quality evaluation standard that defines (a) a 'quality in use model' of five characteristics that relate to the outcome of interaction when a product is used in a particular context of use, and (b) a 'product quality model' composed of eight characteristics that relate to static properties of software and dynamic properties of the computer system.   
  [ISO/IEC/IEEE 29119 series of standards](http://www.softwaretestingstandard.org/) for software testing.   
     ISO/IEC/IEEE 29119-1: Concepts & Definitions   
     ISO/IEC/IEEE 29119-2: Test Processes   
     ISO/IEC/IEEE 29119-3: Test Documentation   
     ISO/IEC/IEEE 29119-4: Test Techniques   
     ISO/IEC/IEEE 29119-5: Keyword Driven Testing
* IEEE = 'Institute of Electrical and Electronics Engineers' - among other things, creates standards such as 'IEEE Standard for Software Test Documentation' (IEEE/ANSI Standard 829), 'IEEE Standard of Software Unit Testing (IEEE/ANSI Standard 1008), 'IEEE Standard for Software Quality Assurance Plans' (IEEE/ANSI Standard 730), and others.
* ANSI = 'American National Standards Institute', the primary industrial standards body in the U.S.; publishes some software-related standards in conjunction with the IEEE and ASQ (American Society for Quality).
* Other software development/IT management process assessment methods besides CMMI and ISO 9000 include SPICE, Trillium, TickIT, Bootstrap, [ITIL](http://www.itil-officialsite.com/home/home.aspx), MOF, and CobiT.
* See the Softwareqatest.com ['Other Resources'](http://www.softwareqatest.com/qatlnks1.html) section for further information available on the web.

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)

**What is the 'software life cycle'?**  
The life cycle begins when an application is first conceived and ends when it is no longer in use. It includes aspects such as initial concept, requirements analysis, functional design, internal design, documentation planning, test planning, coding, document preparation, integration, testing, maintenance, updates, retesting, phase-out, agile sprints, and other aspects. (See the [Softwareqatest.com Bookstore](http://www.softwareqatest.com/qatbks1.html) section's 'Software QA', 'Software Engineering', and 'Project Management' categories for useful books with more information.)

[Return to top of this page's FAQ list](http://www.softwareqatest.com/qatfaq1.html#TOP)