CHAPTER 7

SOFTWARE DEVELOPMENT

QUOTE

Any process that tries to reduce software development to a "no brainer" will eventually produce just that: a product developed by people without brains.
—Andy Hunt and Dave Thomas, "Cook Until Done"

VIGNETTE

Stock Markets Susceptible to Software Glitches

Regulation National Market System (Reg NMS) is a set of rules implemented by the Securities and Exchange Commission in 2007 to boost competition across the various stock exchanges. Reg NMS essentially enables traders to do comparison shopping across the various exchanges to find the best price. The rules also had the effect of lowering trading costs and accelerating the speed of trade executions to a split second.[1] The implementation of Reg NMS led to a rise in the number of firms engaged in "high-frequency trading"—that is, trading that uses powerful computers and complex computer algorithms to trade hundreds or even thousands of times a day. High-frequency trading often employs stock holding periods of only a few seconds to take advantage of tiny price changes.[2] Unfortunately, there have been several recent examples in which problems with the software used in high-frequency trading operations have wreaked havoc on the market—causing problems for the listed companies and stock traders alike.

On May 6, 2010, a "flash crash" of U.S. stock markets occurred in which the Dow Jones Industrial average dropped over 700 points in five minutes, only to recover 600 points over the course of the next 20 minutes. It was a roller coaster ride that briefly erased $1 trillion in market value and left investors and regulators struggling to understand what had happened.[3] Ultimately, it was determined that the flash crash was caused by the actions of a single, large investor who was using automated trading software to trade futures on the stocks in the Standard & Poor 500 stock index. The software placed large sell orders that were, at first, absorbed by other buyers—many of whom were also using automated trading software. However, the algorithm used by the seller's trading software responded to the increase in market activity in the futures contracts by automatically placing larger and larger sell orders, which the market could no longer absorb. This resulted in a rapid decline in the prices of the underlying stocks.[4]

Better Alternative Trading System (BATS) is the third-largest equities exchange operator in the United States. The BATS Exchange accounts for 11 percent of the trading volume of U.S. stock shares.[5,6] On March 23, 2012, the day that BATS launched its own initial public offering (IPO), a "bad trade" for shares of Apple at an incorrect price was accepted on the BATS Exchange. This triggered a flurry of high-frequency trading that resulted in a 9.4 percent drop in Apple's stock's price in just five minutes. The sudden drop in price triggered a trading "circuit breaker" that halted trading in Apple shares and likely prevented a broader crash affecting other stocks. As it turned out, BATS was having technical problems in processing orders for any companies whose ticket symbol was in the range of A to BF, a range that includes not only Apple, but also the firm's own symbol BATS. The BATS stock opened at $16 per share, and at one point appeared to be selling at less than a penny per share. Eventually BATS was forced to halt trading of its own stock and cancel all trades of the stock for that day. BATS also pulled its IPO, which was put on hold "for the foreseeable future."[7,8]

In May 2012, computer problems on the NASDAQ stock exchange disrupted Facebook's IPO. As a result, Facebook trading was delayed half an hour beyond the normal trading time for an IPO. Once trading began, over 80 million shares of Facebook were traded in the first 30 seconds, but traders complained that their orders were not being completed promptly and that they were being charged more than expected. Traders also complained that they were not getting confirmation on their Facebook trades; thus, they did not know if they owned the stock or not.

Knight Capital is a global financial services firm that engages in market making of U.S. securities and electronic stock transaction execution. As a market maker, the firm holds a large quantity of shares of various stocks to facilitate trading in those stocks. The firm displays buy and sell prices it is willing to accept and when an order is received, the market maker immediately fills a buy order from its own inventory or finds a buyer for a sell order. All this happens in seconds. In August 2012, following the installation of new trading software, Knight Capital's computers sent incorrect orders for over 140 stocks listed on the New York Stock Exchange.[9] As a result, several of these stocks traded at 20 times their normal volume and lost over 10 percent of their value in a matter of seconds before recovering.[10] After sorting through the transactions, the New York Stock Exchange canceled trades in the stocks that were most affected by the problem. In the days following the glitch, Knight Capital's own stock price fell from just over $10 per share to under $3 per share.[11]

According to the rules of the stock exchanges, stock orders must be routed to those exchanges that offer the best bid and offer prices. In January 2013, BATS was forced to acknowledge more software-related problems when it notified its clients that due to problems with its trading software, the firm did not meet that requirement for many orders. As a result, over the course of four years, hundreds of thousands of orders were executed at inferior prices.[12] The full fallout from this admission is yet to be seen, but trader lawsuits and other repercussions can be expected.

Questions to Consider

1. Do you think that the software development teams responsible for the development of programs to support high-frequency trading and other sophisticated trading tools should bear any responsibility for these trading fiascos? Why or why not?
2. What measures could be taken to improve the quality of trading software to avoid the problems discussed in the opening vignette?

LEARNING OBJECTIVES

As you read this chapter, consider the following questions:

1. Why must companies place an increased emphasis on the use of high-quality software in business systems, industrial process-control systems, and consumer products?
2. What potential ethical issues do software manufacturers face in making trade-offs between project schedules, project costs, and software quality?
3. What are the four most common types of software product liability claims?
4. What are the essential components of a software development methodology, and what are the benefits of using such a methodology?
5. How can the Capability Maturity Model Integration® improve an organization's software development process?
6. What is a safety-critical system, and what special actions are required during its development?

STRATEGIES FOR ENGINEERING QUALITY SOFTWARE

High-quality software systems are systems that are easy to learn and use because they perform quickly and efficiently; they meet their users' needs; and they operate safely and reliably so that system downtime is kept to a minimum. Such software has long been required to support the fields of air traffic control, nuclear power, automobile safety, health care, military and defense, and space exploration. Now that computers and software have become integral parts of almost every business, the demand for high-quality software is increasing. End users cannot afford system crashes, lost work, or lower productivity. Nor can they tolerate security holes through which intruders can spread viruses, steal data, or shut down Web sites. Software manufacturers face economic, ethical, and organizational challenges associated with improving the quality of their software. This chapter covers many of these issues.

A **software defect** is any error that, if not removed, could cause a software system to fail to meet its users' needs. The impact of these defects can be trivial; for example, a computerized sensor in a refrigerator's ice cube maker might fail to recognize that the tray is full and continue to make ice. Other defects could lead to tragedy—the control system for an automobile's antilock brakes could malfunction and send the car into an uncontrollable spin. The defect might be subtle and undetectable, such as a tax preparation package that makes a minor miscalculation; or the defect might be glaringly obvious, such as a payroll program that generates checks with no deductions for Social Security or other taxes. Here are some notable software bugs that have occurred recently:

- Nokia's Lumina 900 smartphone had a software problem that could cause the device to lose its high-speed data connection. The company had hoped that the new smartphone would help raise its share of the U.S. market, which had slipped below 1 percent. The software glitch was a major setback for the firm and caused it to lower its profit forecasts. As a result, Nokia shares hit a 15-year low.[13]
- The IRS plans to invest $1.3 billion through 2024 to update its software for handling the filing of tax returns; however, an early change designed to speed up the processing of tax returns resulted in refunds that were delayed by up to 10 days for millions of taxpayers in 2012.[14]
- Some 4,000 owners of the 2013 Chevy Volt were informed that a software bug could cause their plug-in hybrid car's electric motor to shut down while the vehicle is in motion.[15]
- In late 2012, many people looking to take part in a Georgia Powerball game were upset when a software error interrupted sales of tickets for the $425 million jackpot. The problem was widespread, affecting many locations in Georgia, lasting most of the day of the drawing.[16]
- Washington State University recently implemented a $15 million software system designed to handle all major student processes—from registering for class, to paying tuition, to scheduling of advisers. Unfortunately, user unfamiliarity with the system and software bugs led to lengthy delays in processing financial aid. Many students who normally rely on financial aid had to dip into reserve funds or call upon parents to help pay for tuition, books, housing, and food until they could receive their financial aid.[17]

Software quality is the degree to which a software product meets the needs of its users. **Quality management** focuses on defining, measuring, and refining the quality of the development process and the products developed during its various stages. These products—including statements of requirements, flowcharts, and user documentation—are known as **deliverables**. The objective of quality management is to help developers deliver high-quality systems that meet the needs of their users. Unfortunately, the first release of any software rarely meets all its users' expectations. A software product does not usually work as well as its users would like it to until it has been used for a while, found lacking in some ways, and then corrected or upgraded.

A primary cause of poor software quality is that many developers do not know how to design quality into software from the very start; some simply do not take the time to do so. To develop high-quality software, developers must define and follow a set of rigorous software engineering principles and be committed to learning from past mistakes. In addition, they must understand the environment in which their systems will operate and design systems that are as immune to human error as possible.

All software designers and programmers make mistakes in defining user requirements and turning them into lines of code. According to one study, even experienced software developers unknowingly inject an average of one design or implementation defect for every 7 to 10 lines of code. The developers aren't incompetent or lazy—they're just human. Everyone makes mistakes, but in software, these mistakes can result in defects.

Based on an analysis of a sample of 300 million lines of commercial code, Coverity (a software development testing firm) found that the average number of defects per thousand lines of code developed by software manufacturing companies was .64.[18] The Microsoft Windows 7 operating system contains more than 50 million lines of code. Assuming the Microsoft software developers produced code at this accuracy rate, there would still be roughly 32,000 defects in Windows 7. Thus, critical software used daily by workers worldwide likely contains tens of thousands of defects. Interestingly, based on an analysis of 37 million lines of open source code, Coverity found that the average number of defects per thousand lines of code was .45 or 30% less than in commercial code.[19]

Another factor that can contribute to poor-quality software is the extreme pressure that software companies feel to reduce the time to market for their products. They are driven by the need to beat the competition in delivering new functionality to users, to begin generating revenue to recover the cost of development, and to show a profit for shareholders. They are also driven by the need to meet quarterly earnings forecasts used by financial analysts to place a value on the stock. The resources and time needed to ensure quality are often cut under the intense pressure to ship a new product. When forced to choose between adding more user features and doing more testing, most software companies decide in favor of more features. They often reason that defects can be patched in the next release, which will give customers an automatic incentive to upgrade. Additional features make a release more useful and therefore easier to sell to customers. A major ethical dilemma for software development organizations is: "How much additional cost and effort should they expend to ensure that their products and services meet customers' expectations?" Over 1.25 million apps have been created for the various types of mobile devices; however, it is estimated that less than 20 percent of these apps exceed 1,000 downloads because of faulty software quality that results in poor application performance.[20] Customers are stakeholders who are key to the success of a software application, and they may benefit from new features. However, they also bear the burden of errors that aren't caught or fixed during testing. Thus, customers challenge whether to cut software quality in favor of feature enhancement.

As a result of the lack of consistent quality in software, many organizations avoid buying the first release of a major software product or prohibit its use in critical systems; their rationale is that the first release often has many defects that cause problems for users. Because of the defects in the first two popular Microsoft operating systems (DOS and Windows), including their tendency to crash unexpectedly, many believe that Microsoft did not have a reasonably reliable operating system until its third major variation—Windows NT.

Even software products that have been reliable over a long period can falter unexpectedly when operating conditions change. For instance, software in the Cincinnati Bell telephone switch had been thoroughly tested and had operated successfully for months after it was deployed. However, when the time changed from daylight saving time to standard time for the first time after the software was deployed, the switch failed because it was overwhelmed by the number of calls to the local "official time" phone number from people who wanted to set their clocks. The large increase in the number of simultaneous calls to the same number was a change in operating conditions that no one had anticipated.

The Importance of Software Quality

A **business information system** is a set of interrelated components—including hardware, software, databases, networks, people, and procedures—that collects and processes data and disseminates the output. A common type of business system is one that captures and records business transactions. For example, a manufacturer's order-processing system captures order information, processes it to update inventory and accounts receivable, and ensures that the order is filled and shipped on time to the customer. Other examples include an airline's online ticket-reservation system and an electronic funds transfer system that moves money among banks. The accurate, thorough, and timely processing of business transactions is a key requirement for such systems. A software defect can be devastating, resulting in lost customers and reduced revenue. How many times would bank customers tolerate having their funds transferred to the wrong account before they stopped doing business with that bank?

Another type of business information system is the **decision support system (DSS)**, which is used to improve decision making in a variety of industries. A DSS can be used to develop accurate forecasts of customer demand, recommend stocks and bonds for an investment portfolio, or schedule shift workers in such a way as to minimize cost while meeting customer service goals. A software defect in a DSS can result in significant negative consequences for an organization and its customers.

Software is also used to control many industrial processes in an effort to reduce costs, eliminate human error, improve quality, and shorten the time it takes to manufacture products. For example, steel manufacturers use process-control software to capture data from sensors about the equipment that rolls steel into bars and about the furnace that heats the steel before it is rolled. Without process-control computers, workers could react to defects only after the fact and would have to guess at the adjustments needed to correct the process. Process-control computers enable the process to be monitored for variations from operating standards (e.g., a low furnace temperature or incorrect levels of iron ore) and to eliminate product defects before they affect product quality. Any defect in this software can lead to decreased product quality, increased waste and costs, or even unsafe operating conditions for employees.

Software is also used to control the operation of many industrial and consumer products, such as automobiles, medical diagnostic and treatment equipment, televisions, radios, stereos, refrigerators, and washers. A software defect could have relatively minor consequences, such as clothes not drying long enough, or it could cause serious damage, such as a patient being overexposed to powerful X-rays.

As a result of the increasing use of computers and software in business, many companies are now in the software business whether they like it or not. The quality of software, its usability, and its timely development are critical to almost everything businesses do. The speed with which an organization develops software can put it ahead of or behind its competitors. Software problems may have caused frustrations in the past, but mismanaged software can now be fatal to a business, causing it to miss product delivery dates, incur increased product development costs, and deliver products that have poor quality.

Business executives frequently face ethical questions of how much money and effort they should invest to ensure the development of high-quality software. A manager who takes a short-term, profit-oriented view may feel that any additional time and money spent on quality assurance will only delay a new product's release, resulting in a delay in sales revenue and profits. However, a different manager may consider it unethical not to fix all known problems before putting a product on the market and charging customers for it.

Other key questions for executives are whether their products could cause damage and what their legal exposure would be if they did. Fortunately, software defects are rarely lethal, and few personal injuries are related to software failures. However, the use of software introduces product liability issues that concern many executives.

SOFTWARE PRODUCT LIABILITY

Software product litigation is certainly not new. One lawsuit in the early 1990s involved a financial institution that became insolvent because defects in a purchased software application caused errors in its integrated general ledger system, customers' passbooks, and loan statements. Dissatisfied depositors responded by withdrawing more than $5 million. Another case involved an accident that occurred when a Ford truck stalled because of a software defect in the truck's fuel injector. In the ensuing accident, a young child was killed.[21] A state supreme court later affirmed an award of $7.5 million in punitive damages against the manufacturer. In 2008, a faulty onboard computer caused a Qantas passenger flight traveling between Perth and Singapore to plunge some 8,000 feet in 10 seconds, injuring 46 passengers. Qantas moved quickly to compensate all passengers with a refund of their ticket prices, a $2,000 travel voucher, and a promise to pay all medical-related expenses. Even so, the Australian law firm of Slater & Gordon was engaged to represent a dozen of the passengers.[22]

The liability of manufacturers, sellers, lessors, and others for injuries caused by defective products is commonly referred to as **product liability**. There is no federal product liability law; instead, product liability in the United States is mainly covered by common law (made by state judges) and Article 2 of the Uniform Commercial Code, which deals with the sale of goods.

If a software defect causes injury or loss to purchasers, lessees, or users of the product, the injured parties may be able to sue as a result. Injury or loss can come in the form of physical mishaps and death, loss of revenue, or an increase in expenses due to a business disruption caused by a software failure. Software product liability claims are typically based on strict liability, negligence, breach of warranty, or misrepresentation—sometimes in combination with one another.

Strict liability means that the defendant is held responsible for injuring another person, regardless of negligence or intent. The plaintiff must prove only that the software product is defective or unreasonably dangerous and that the defect caused the injury.

There is no requirement to prove that the manufacturer was careless or negligent, or to prove who caused the defect. All parties in the chain of distribution—the manufacturer, subcontractors, and distributors—are strictly liable for injuries caused by the product and may be sued.

Defendants in a strict liability action may use several legal defenses, including the doctrine of supervening event, the government contractor defense, and an expired statute of limitations. Under the doctrine of supervening event, the original seller is not liable if the software was materially altered after it left the seller's possession and the alteration caused the injury. To establish the government contractor defense, a contractor must prove that the precise software specifications were provided by the government, that the software conformed to the specifications, and that the contractor warned the government of any known defects in the software. Finally, there are statutes of limitations for claims of liability, which means that an injured party must file suit within a certain amount of time after the injury occurs.

As discussed in Chapter 2, negligence is the failure to do what a reasonable person would do, or doing something that a reasonable person would not do. When sued for negligence, a software supplier is not held responsible for every product defect that causes customer or third-party loss. Instead, responsibility is limited to harmful defects that could have been detected and corrected through "reasonable" software development practices. Contracts written expressly to limit claims of supplier negligence may be disregarded by the courts as unreasonable. Software manufacturers or organizations with software-intensive products are frequently sued for negligence and must be prepared to defend themselves.

The defendant in a negligence case may either answer the charge with a legal justification for the alleged misconduct or demonstrate that the plaintiffs' own actions contributed to their injuries (**contributory negligence**). If proved, the defense of contributory negligence can reduce or totally eliminate the amount of damages the plaintiffs receive. For example, if a person uses a pair of pruning shears to trim his fingernails and ends up cutting off a fingertip, the defendant could claim contributory negligence.

A **warranty** assures buyers or lessees that a product meets certain standards of quality. A warranty of quality may be either expressly stated or implied by law. Express warranties can be oral, written, or inferred from the seller's conduct. For example, sales contracts contain an implied warranty of merchantability, which requires that the following standards be met:

- The goods must be fit for the ordinary purpose for which they are used.
- The goods must be adequately contained, packaged, and labeled.
- The goods must be of an even kind, quality, and quantity within each unit.
- The goods must conform to any promise or affirmation of fact made on the container or label.
- The quality of the goods must pass without objection in the trade.
- The goods must meet a fair average or middle range of quality.

If the product fails to meet the terms of its warranty, the buyer or lessee can sue for **breach of warranty**. Of course, most dissatisfied customers will first seek a replacement, a substitute product, or a refund before filing a lawsuit.

Software suppliers frequently write warranties to attempt to limit their liability in the event of nonperformance. Although a certain software may be warranted to run on a given machine configuration, often no assurance is given as to what that software will do. Even if a contract specifically excludes the commitment of merchantability and fitness for a specific use, the court may find such a disclaimer clause unreasonable and refuse to enforce it or refuse to enforce the entire contract. In determining whether a warranty disclaimer is unreasonable, the court attempts to evaluate if the contract was made between two "equals" or between an expert and a novice. The relative education, experience, and bargaining power of the parties and whether the sales contract was offered on a "take-it-or-leave-it" basis are considered in making this determination.

The plaintiff must have a valid contract that the supplier did not fulfill in order to win a breach-of-warranty claim. Because the software supplier writes the warranty, this claim can be extremely difficult to prove. For example, the M. A. Mortenson Company—one of the largest construction companies in the United States—installed a new version of bid-preparation software for use by its estimators. During the course of preparing one new bid, the software allegedly malfunctioned several times, each time displaying the same cryptic error message. Nevertheless, the estimator submitted the bid and Mortenson won the contract. Afterward, Mortenson discovered that the bid was $1.95 million lower than intended, and the company filed a breach-of-warranty suit against Timberline Software, makers of the bid software. Timberline acknowledged the existence of the bug. However, the courts ruled in Timberline's favor because the license agreement that came with the software explicitly barred recovery of the losses claimed by Mortenson.[23] Even if breach of warranty can be proven, the damages are generally limited to the amount of money paid for the product.

As mentioned in Chapter 2, intentional misrepresentation occurs when a seller or lessor either misrepresents the quality of a product or conceals a defect in it. For example, if a cleaning product is advertised as safe to use in confined areas and some users subsequently pass out from the product's fumes, they could sue the seller for intentional misrepresentation or fraud. Advertising, salespersons' comments, invoices, and shipping labels are all forms of representation. Most software manufacturers use limited warranties and disclaimers to avoid any claim of misrepresentation.

Software Development Process

Developing information system software is not a simple process; it requires completing many complex activities, with many dependencies among the various activities. Systems analysts, programmers, architects, database specialists, project managers, documentation specialists, trainers, and testers are all involved in large software projects. Each of these groups of workers has a role to play and has specific responsibilities and tasks. In addition, each group makes decisions that can affect the software's quality and the ability of an organization or an individual to use it effectively.

Most software companies have adopted a **software development methodology**—a standard, proven work process that enables systems analysts, programmers, project managers, and others to make controlled and orderly progress while developing high-quality software. A methodology defines activities in the software development process and the individual and group responsibilities for accomplishing these activities. It also recommends specific techniques for accomplishing the various activities, such as using a flowchart to document the logic of a computer program. A methodology also offers guidelines for managing the quality of software during the various stages of development. See Figure 7-1. If an organization has developed such a methodology, it is typically applied to any software development that the company undertakes.

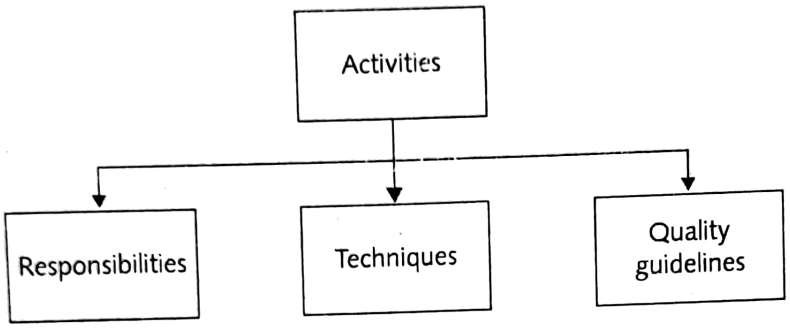

FIGURE 7-1 Software development methodology
Source Line: Course Technology/Cengage Learning.

As with most things, it is usually easier and cheaper to avoid software problems from the beginning, rather than attempt to fix the damages after the fact. Studies have shown that the cost to identify and remove a defect in an early stage of software development (requirements definition) can be up to 100 times less than removing a defect in a piece of software that has been distributed to customers (see Figure 7-2).[24,25] Although these studies were conducted several years ago, their results still hold true today.

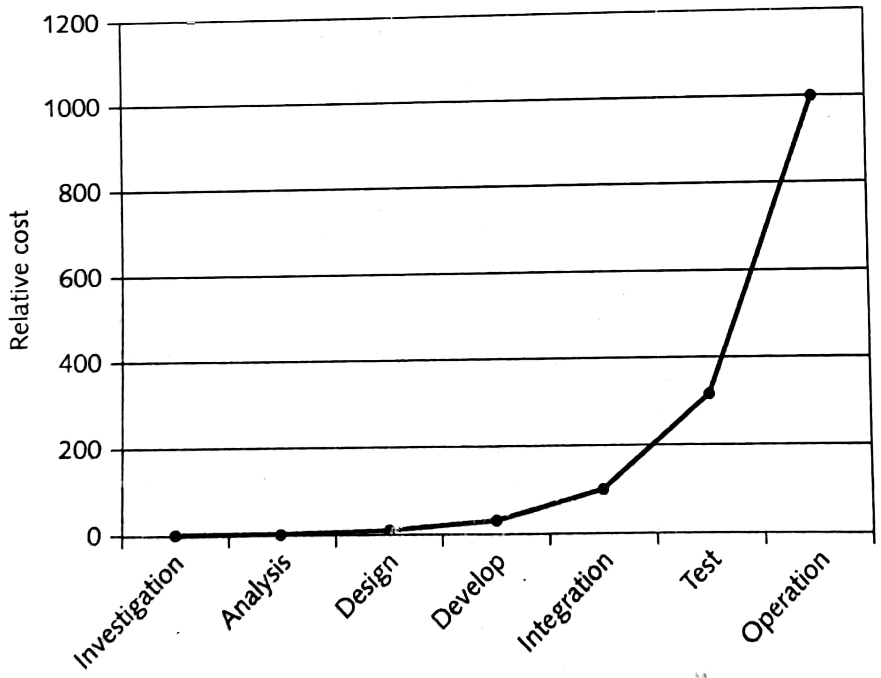

FIGURE 7-2 The cost of removing software defects
Source Line: Used with permission from LKP Consulting Group.

If a defect is uncovered during a later stage of development, some rework of the deliverables produced in preceding stages will be necessary. The later the error is detected, the greater the number of people who will be affected by the error. Thus, the greater the costs will be to communicate and fix the error. Consider the cost to communicate the details of a defect, distribute and apply software fixes, and possibly retrain end users for a software product that has been sold to hundreds or thousands of customers. Thus, most software developers try to identify and remove errors early in the development process not only as a cost-saving measure but also as the most efficient way to improve software quality.

A product containing inherent defects that harm the user may be the subject of a product liability suit. The use of an effective methodology can protect software manufacturers from legal liability in two ways. First, an effective methodology reduces the number of software errors that might occur. Second, if an organization follows widely accepted development methods, negligence on its part is harder to prove. However, even a successful defense against a product liability case can cost hundreds of thousands of dollars in legal fees. Thus, failure to develop software carefully and consistently can be serious in terms of liability exposure.

Quality assurance (QA) refers to methods within the development cycle designed to guarantee reliable operation of a product. Ideally, these methods are applied at each stage of the development cycle. However, some software manufacturing organizations without a formal, standard approach to QA consider testing to be their only QA method. Instead of checking for errors throughout the development process, such companies rely primarily on testing just before the product ships to ensure some degree of quality.

Several types of tests are used in software development, as discussed in the following sections.

Dynamic Software Testing

Software is developed in units called subroutines or programs. These units, in turn, are combined to form large systems. One approach to QA is to test the code for a completed unit of software by actually entering test data and comparing the results with the expected results in a process called **dynamic testing**. There are two forms of dynamic testing:

- **Black-box testing** involves viewing the software unit as a device that has expected input and output behaviors but whose internal workings are unknown (a black box). If the unit demonstrates the expected behaviors for all the input data in the test suite, it passes the test. Black-box testing takes place without the tester having any knowledge of the structure or nature of the actual code. For this reason, it is often done by someone other than the person who wrote the code.
- **White-box testing** treats the software unit as a device that has expected input and output behaviors but whose internal workings, unlike the unit in black-box testing, are known. White-box testing involves testing all possible logic paths through the software unit with thorough knowledge of its logic. The test data must be carefully constructed so that each program statement executes at least once. For example, if a developer creates a program to calculate an

employee's gross pay, the tester would develop data to test cases in which the employee worked less than 40 hours, exactly 40 hours, and more than 40 hours (to check the calculation of overtime pay).

Other Types of Software Testing

Other forms of software testing include the following:

- *Static testing*—Special software programs called static analyzers are run against new code. Rather than reviewing input and output, the static analyzer looks for suspicious patterns in programs that might indicate a defect.
- *Integration testing*—After successful unit testing, the software units are combined into an integrated subsystem that undergoes rigorous testing to ensure that the linkages among the various subsystems work successfully.
- *System testing*—After successful integration testing, the various subsystems are combined to test the entire system as a complete entity.
- *User acceptance testing*—Independent testing is performed by trained end users to ensure that the system operates as they expect.

Capability Maturity Model Integration

Capability Maturity Model Integration (CMMI)—developed by the Software Engineering Institute at Carnegie Mellon—is a process-improvement approach that defines the essential elements of effective processes. The model is general enough to be used to evaluate and improve almost any process, and a specific application of CMMI—**CMMI-Development (CMMI-DEV)**—is frequently used to assess and improve software development practices. CMMI defines five levels of software development maturity (see Table 7-1) and identifies the issues that are most critical to software quality and process improvement.

TABLE 7-1 Definition of CMMI maturity levels

Maturity level	Description
Initial	Process is ad hoc and chaotic; organization tends to overcommit and processes are often abandoned during times of crisis.
Managed	Projects employ processes and skilled people; status of work products is visible to management at defined points.
Defined	Processes are well defined and understood and are described in standards, procedures, tools, and methods; processes are consistent across the organization.
Quantitatively managed	Quantitative objectives for quality and process performance are established and are used as criteria in managing projects; specific measures of process performance are collected and statistically analyzed.
Optimizing	Organization continually improves its processes; changes are based on a quantitative understanding of its business objectives and performance needs.

Source Line: Used with permission from Carnegie Mellon University.

A maturity level consists of practices for a set of process areas that improve an organization's overall performance. Identifying an organization's current maturity level enables it to specify necessary actions to improve the organization's future performance. The model also enables an organization to track, evaluate, and demonstrate its progress over the years.

Table 7-2 shows the percentages of organizations at each CMMI maturity level, as reported in a recent survey of 5,159 reporting organizations as of September 2012.

TABLE 7-2 Maturity level distribution across a large sample of organizations

Maturity level	Percent of 5,159 organizations surveyed
Not provided	3.3%
Initial	0.8%
Managed	22.1%
Defined	66.8%
Quantitatively managed	1.6%
Optimizing	6.5%

Source Line: Used with permission from Carnegie Mellon University.

The Software Engineering Institute documented the following results from CMMI process-improvement implementations at 11 different organizations:

- A 33 percent decrease in the cost to fix defects
- A 20 percent reduction in unit software costs
- A 30 percent increase in productivity
- An increase in project-schedule milestones met—from 50 percent to 95 percent[26]

CMMI-DEV is a set of guidelines for 22 process areas related to systems development. The premise of the model is that those organizations that do these 22 things well will have an outstanding software development process. After an organization decides to adopt CMMI-DEV, it must conduct an assessment of its software development practices (using trained, outside assessors to ensure objectivity) to determine where the organization fits in the capability model. The assessment identifies areas for improvement and establishes action plans needed to upgrade the development process. Over the course of a few years, the organization can improve its maturity level by executing the action plan.

CMMI-DEV can also be used as a benchmark for comparing organizations. In the awarding of software contracts—particularly by the federal government—organizations that bid on a contract may be required to have adopted CMMI and to be performing at a certain level.

Achieving Maturity Level 5—the highest possible rating—is a significant accomplishment for any organization, and it can lead to substantial business benefits. It means that the organization is able to statistically evaluate the performance of its software development processes. This in turn leads to better control and continual improvement in the processes, making it possible to deliver software products of high quality on time and on budget.

At the Rolling Meadows campus of Northrop Grumman, workers design, develop, and manufacture advanced electronic systems for customers worldwide. The campus was recently assessed and rated at CMMI Level 5. According to Dan Blase, director of engineering at the facility, "Continuous improvement is an integral part of the culture at Northrop Grumman. This CMMI rating reflects our commitment to performing at the highest level for our customers, and doing so in the most affordable manner possible."[27]

KEY ISSUES IN SOFTWARE DEVELOPMENT

Although defects in any system can cause serious problems, the consequences of software defects in certain systems can be deadly. In these kinds of systems, the stakes involved in creating quality software are raised to the highest possible level. The ethical decisions involving a trade-off—if one must be considered—between quality and such factors as cost, ease of use, and time to market require extremely serious examination. The next sections discuss safety-critical systems and the special precautions companies must take in developing them.

Development of Safety-Critical Systems

A safety-critical system is one whose failure may cause human injury or death. The safe operation of many safety-critical systems relies on the flawless performance of software; such systems control automobiles' antilock brakes, nuclear power plant reactors, airplane navigation, elevators, and numerous medical devices, to name just a few. The process of building software for such systems requires highly trained professionals, formal and rigorous methods, and state-of-the-art tools. Failure to take strong measures to identify and remove software errors from safety-critical systems "is at best unprofessional and at worst lead[s] to disastrous consequences."[28] However, even with these types of precautions, the software associated with safety-critical systems is still vulnerable to errors that can lead to injury or death. Here are several examples of safety-critical system failures:

- The *Mariner I* space probe, which was intended to make a close flyby of the planet Venus, was ordered destroyed less than five minutes after launch in July 1962. Faulty software code caused the flight control computer to perform a series of unnecessary course corrections, which threw the spacecraft dangerously off course.[29]
- A Royal Air Force helicopter took off from Northern Ireland in June 1994 with 25 British intelligence officials who were heading to a security conference in Inverness. Just 18 minutes into its flight, the helicopter crashed on the peninsula of Kintyre in Argyll, Scotland, killing everyone on board. The engine management software, which controlled the acceleration and deceleration of the engines, was suspected of causing the crash.[30]
- Between November 2000 and March 2002, therapy planning software at the National Oncology Institute in Panama City, Panama, miscalculated the proper dosage of radiation for patients undergoing therapy; at least eight patients died while another 20 received overdoses that caused significant health problems.[31]

- Three accidents occurred on the Big Thunder Mountain Railroad roller coaster at Disneyland between September 2003 and July 2004. One person was killed and 10 others were injured in the September accident. The California Division of Occupational Safety and Health blamed the accidents on improper maintenance, poorly trained operators, and a glitch in the ride's computer system.[32]
- In April 2007, fire broke out on a Washington, D.C., six-car Metro train as it pulled out of the L'Enfant Plaza station. Fire and smoke were seen underneath the last car, but thankfully, the flames did not penetrate the floor of the car. The train operator stopped and evacuated the passengers. It was eventually determined that the train's brake resistor grid, which checks various subsystems and voltages, overheated and caught fire. Monitoring software failed to perform as expected in detecting and preventing excess power usage in equipment on the passenger rail cars, resulting in overheating and fire.[33]

When developing safety-critical systems, a key assumption must be that safety will *not* automatically result from following an organization's standard development methodology. Safety-critical software must go through a much more rigorous and time-consuming development process than other kinds of software. All tasks—including requirements definition, systems analysis, design, coding, fault analysis, testing, implementation, and change control—require additional steps, more thorough documentation, and vigilant checking and rechecking. As a result, safety-critical software takes much longer to complete and is much more expensive to develop.

Software developers working on a safety-critical system must also recognize that the software is only one component of the system; other components typically include system users or operators, hardware, and other equipment. Software developers must work closely with safety and systems engineers to ensure that the entire system, not just the software, operates in a safe manner.

The key to ensuring that these additional tasks are completed is to appoint a **system safety engineer**, who has explicit responsibility for the system's safety. The safety engineer uses a logging and monitoring system to track hazards from a project's start to its finish. This **hazard log** is used at each stage of the software development process to assess how it has accounted for detected hazards. Safety reviews are held throughout the development process, and a robust configuration management system tracks all safety-related matters. However, the safety engineer must keep in mind that his or her role is not simply to produce a hazard log but rather to influence the design of the system to ensure that it operates safely when put into use.

The increased time and expense of completing safety-critical software can draw developers into ethical dilemmas. For example, the use of hardware mechanisms to back up or verify critical software functions can help ensure safe operation and make the consequences of software defects less critical. However, such hardware may make the final product more expensive to manufacture or harder for the user to operate—potentially making the product less attractive than a competitor's. Companies must carefully weigh these issues to develop the safest possible product that also appeals to customers.

Another key issue is deciding when the QA staff has performed sufficient testing. How much testing is enough when you are building a product whose failure could cause loss of human life? At some point, software developers must determine that they have completed sufficient QA activities and then sign off to indicate their approval. Determining how much testing is sufficient demands careful decision making.

When designing, building, and operating a safety-critical system, a great deal of effort must be put into considering what can go wrong, the likelihood and consequences of such occurrences, and how risks can be averted, mitigated, or detected so the users can be warned. One approach to answering these questions is to conduct a formal risk analysis. **Risk** is the probability of an undesirable event occurring times the probability that the event would go undetected times the magnitude of the event's consequences if it does happen. These consequences include damage to property, loss of money, injury to people, and death.

For example, if an undesirable event has a 1 percent probability of occurring, a 25 percent chance of going undetected, and a potential cost of $1,000,000, then the risk can be calculated as 0.01 × .25 × $1,000,000, or $2,500. The risk for this event would be considered greater than that of an event with a 10 percent probability of occurring, a 20 percent chance of going undetected, and a potential cost of $100 (0.10 × .20 × $100 = $2.00). Risk analysis is important for safety-critical systems but is useful for other kinds of software development as well.

Another key element of safety-critical systems is **redundancy**, the provision of multiple interchangeable components to perform a single function in order to cope with failures and errors. An example of a simple redundant system would be an automobile with a spare tire or a parachute with a backup chute attached. A more complex system used in IT is a redundant array of independent disks (RAID), which is commonly used in high-volume data storage for file servers. RAID systems use many small-capacity disk drives to store large amounts of data to provide increased reliability and redundancy. Should one of the drives fail, it can be removed and a new one inserted in its place. Because the data has also been stored elsewhere, data on the failed disk can be rebuilt automatically without the server ever having to be shut down.

During times of widespread disaster, lack of sufficient redundant systems can lead to major problems. For example, the designers of the reactors at Japan's Fukushima Daiichi Nuclear Power Plant anticipated that a strong earthquake and even a tsunami might hit the facility. So in addition to a main power supply, backup generators were put in place to ensure that coolant could be circulated to the nuclear reactors even if the main power supply was knocked out. When a 9.0 earthquake hit the area in early 2011, it knocked out the main power supply, but the backup power supply was still working until it was hit with a tsunami 10 meters high, twice the height of what had been anticipated in the design of the redundant power supplies of the plant.[34]

N-version programming is an approach to minimizing the impact of software errors by independently implementing the same set of user requirements *N* times (where *N* could be 2, 3, 4, or more). The different versions of the software are run in parallel, and if the outputs of the different software vary, a "voting algorithm" is executed to determine which result to use. For example, if two software versions calculate the answer to be 2.4 and the third version calculates 4.1, the algorithm might choose 2.4 as the correct answer. Each software version is built by different teams of people using different approaches to write programming instructions designed to meet the users' requirements. In some cases, instructions are written by teams of programmers from different companies and run on different hardware devices. The rationale behind N-version programming is that multiple software versions are highly unlikely to fail at the same time under the same conditions. Thus, one or more of the versions should yield a correct result. Triple-version programming is common in airplane and spacecraft control systems.

After an organization determines all pertinent risks to a system, it must decide what level of risk is acceptable. This decision is extremely difficult and controversial because it involves forming personal judgments about the value of human life, assessing potential liability in case of an accident, evaluating the surrounding natural environment, and estimating the system's costs and benefits. System modifications must be made if the level of risk in the design is judged to be too great. Modifications can include adding redundant components or using safety shutdown systems, containment vessels, protective walls, or escape systems. Another approach is to mitigate the consequences of failure by devising emergency procedures and evacuation plans. In all cases, organizations must ask how safe is safe enough if human life is at stake.

Manufacturers of safety-critical systems must sometimes decide whether to recall a product when data indicates a problem. For example, automobile manufacturers have been known to weigh the cost of potential lawsuits against that of a recall. Drivers and passengers in affected automobiles (and, in many cases, the courts) have not found this approach to be ethically sound. Manufacturers of medical equipment and airplanes have had to make similar decisions, which can be complicated if data cannot pinpoint the cause of a particular problem. For example, there was great controversy in 2000 over the use of Firestone tires on Ford Explorers after numerous tire blowouts and Explorer rollovers caused multiple injuries and deaths. However, it was difficult to determine if the rollovers were caused by poor automobile design, faulty tires, or improperly inflated tires. Consumers' confidence in both manufacturers and their products was nevertheless shaken.

Reliability is a measure of the rate of failure in a system that would render it unusable over its expected lifetime. For example, if a component has a reliability of 99.9 percent, it has one chance in a thousand of failing over its lifetime. Although this chance of failure may seem low, remember that most systems are made up of many components. As you add more components, the system becomes more complex, and the chance of failure increases. For example, assume that you are building a complex system made up of seven components, each with 99 percent reliability. If none of the components has redundancy built in, the system has a 93.8 percent ($.99^7$) probability of operating successfully with no component malfunctions over its lifetime. If you build the same type of system using 10 components, each with 99 percent reliability, the overall probability of operating without an individual component failure falls to 90 percent. Thus, building redundancy into systems that are both complex and safety critical is imperative. System engineers sometimes refer to the goal of designing a system with "five nines" (99.999%) reliability. This translates to a system that would have only 5.26 minutes of total downtime in a year.

Reliability and safety are two different system characteristics. Reliability has to do with the capability of the system to continue to perform; safety has to do with the ability of the system to perform in a safe manner. Thus, a system could be reliable but not safe. For example, an antiaircraft missile control system may continue to operate under a wide range of operating conditions so that it is considerable reliable. If, however, the control system directs the missile to change direction and to fly back into its launching device, it is certainly unsafe.

One of the most important and difficult areas of safety-critical system design is the system-human interface. Human behavior is not nearly as predictable as the performance of hardware and software components in a complex system. The system designer must consider what human operators might do to make a system work less safely or effectively. The challenge is to design a system that works as it should and leaves little room for erroneous judgment on the part of the operator. For instance, a self-medicating pain-relief system must allow a patient to press a button to receive more pain reliever, but must also regulate itself to prevent an overdose. Additional risk can be introduced if a designer does not anticipate the information an operator needs and how the operator will react under the daily pressures of actual operation, especially in a crisis. Some people keep their wits about them and perform admirably in an emergency, but others may panic and make a bad situation worse.

Poor design of a system interface can greatly increase risk, sometimes with tragic consequences. For example, in July 1988, the guided missile cruiser USS *Vincennes* mistook an Iranian Air commercial flight for an enemy F-14 jet fighter and shot the airliner down over international waters in the Persian Gulf. All 290 people on board were killed. Some investigators blamed the tragedy on a lack of training and experience on the part of the operators and the confusing interface of the $500 million Aegis radar and weapons control system. The Aegis radar on the *Vincennes* locked onto an Airbus 300, but it was misidentified as a much smaller F-14 by its human operators. The Aegis operators also misinterpreted the system signals and thought that the target was descending, even though the airbus was actually climbing. A third human error was made in determining the target altitude—it was off by 4,000 feet. As a result of this combination of human errors, the *Vincennes* crew thought the ship was under attack and shot down the plane.[35]

Quality Management Standards

The International Organization for Standardization (ISO), founded in 1947, is a worldwide federation of national standards bodies from 161 countries. The organization issued its 9000 series of business management standards in 1988. These standards require organizations to develop formal quality management systems that focus on identifying and meeting the needs, desires, and expectations of their customers.

The **ISO 9001 family of standards** serves as a guide to quality products, services, and management. ISO 9001:2008 provides a set of standardized requirements for a quality management system. It is the only standard in the ISO 9001 family for which organizations can be certified. Over 1 million organizations in more than 175 countries have ISO 9001 certification.[36] Although companies can use the standard as a management guide for their own purposes in achieving effective control, the priority for many companies is having a qualified external agency certify that they have achieved ISO 9001 certification. Many businesses and government agencies both in the United States and abroad insist that a potential vendor or business partner have a certified quality management system in place as a condition of doing business. Becoming ISO 9001 certified provides proof of an organization's commitment to quality management and continuous improvement.

To obtain this coveted certificate, an organization must submit to an examination by an external assessor and must fulfill the following requirements:

- Have written procedures for all processes
- Follow those procedures

- Prove to an auditor that it has fulfilled the first two requirements; this proof can require observation of actual work practices and interviews with customers, suppliers, and employees

Many software development organizations are applying ISO 9001 to meet the special needs and requirements associated with the purchase, development, operation, maintenance, and supply of computer software.

Failure mode and effects analysis (FMEA) is an important technique used to develop ISO 9001-compliant quality systems by both evaluating reliability and determining the effects of system and equipment failures. Failures are classified according to their impact on a project's success, personnel safety, equipment safety, customer satisfaction, and customer safety. The goal of FMEA is to identify potential design and process failures early in a project, when they are relatively easy and inexpensive to correct.

A **failure mode** describes how a product or process could fail to perform the desired functions described by the customer. An effect is an adverse consequence that the customer might experience. Unfortunately, most systems are so complex that there is seldom a one-to-one relationship between cause and effect. Instead, a single cause may have multiple effects, and a combination of causes may lead to one effect or multiple effects. It is not uncommon for a FMEA of a system to identify 50 to 200 potential failure modes.

The use of FMEA helps to prioritize those actions necessary to reduce potential failures with the highest relative risks. The following steps are used to identify the highest priority actions to be taken:

- *Determine the severity rating*—The potential effects of a failure are scored on a scale of 1 to 10 (or 1 to 5) with 10 assigned to the most serious consequence (9 or 10 are assigned to safety- or regulatory-related effects).
- *Determine the occurrence rating*—The potential causes of that failure occurring are also scored on a scale of 1 to 10, with 10 assigned to the cause with the greatest probability of occurring.
- *Determine the criticality*—Criticality is the product of severity times occurrence.
- *Determine the detection rating*—The ability to detect the failure in advance of it occurring due to the specific cause under consideration is also scored on a scale of 1 to 10, with 10 assigned to the failure with the least likely chance of advance detection. For software, the detection rating would represent the ability of planned tests and inspections to remove the cause of a failure.
- *Calculate the risk priority rating*—The severity rating is multiplied by the occurrence rating and by the detection rating to arrive at the risk priority rating.[37]

Raytheon is a technology company that designs and manufactures aerospace and defense systems that incorporate the latest electronic components. It employs 68,000 people worldwide and generated $24 billion in recent sales.[38] Raytheon employs FMEA throughout its product development life cycle. Starting early in the design cycle, the firm invites suppliers to review its designs to identify potential failure modes, assess the ability to detect the modes, and estimate the severity of the effects. The firm then uses this input to prioritize the product design issues that need to be eliminated or mitigated to create superior products.[39]

Table 7-3 shows a sample FMEA risk priority table.

TABLE 7-3 Sample FMEA risk priority table

Issue	Severity	Occurrence	Criticality	Detection	Risk priority
#1	3	4	12	9	108
#2	9	4	36	2	72
#3	4	5	20	4	80

Source Line: Course Technology/Cengage Learning.

Many organizations consider those issues with the highest criticality rating (severity × occurrence) as the highest priority issues to address. They may then go on to address those issues with the highest risk priority (severity × occurrence × detection). So although Issue #2 shown in Table 7-3 has the lowest risk priority, it may be assigned the highest priority because of its high criticality rating.

Table 7-4 provides a manager's checklist for upgrading the quality of the software an organization produces. The preferred answer to each question is *yes*.

TABLE 7-4 Manager's checklist for improving software quality

Question	Yes	No
Has senior management made a commitment to develop quality software?		
Have you used CMMI to evaluate your organization's software development process?		
Has your company adopted a standard software development methodology?		
Does the methodology place a heavy emphasis on quality management and address how to define, measure, and refine the quality of the software development process and its products?		
Are software project managers and team members trained in the use of this methodology?		
Are software project managers and team members held accountable for following this methodology?		
Is a strong effort made to identify and remove errors as early as possible in the software development process?		
Are both static and dynamic software testing methods used?		
Are white-box testing and black-box testing methods used?		
Has an honest assessment been made to determine if the software being developed is safety critical?		
If the software is safety critical, are additional tools and methods employed, and do they include the following: a project safety engineer, hazard logs, safety reviews, formal configuration management systems, rigorous documentation, risk analysis processes, and the FMEA technique?		

Source Line: Course Technology/Cengage Learning.

Summary

- High-quality software systems are easy to learn and use. Such systems perform quickly and efficiently to meet their users' needs, operate safely and reliably, and have a high degree of availability that keeps unexpected downtime to a minimum.
- High-quality software has long been required to support the fields of air traffic control, nuclear power, automobile safety, health care, military and defense, and space exploration, among others.
- Now that computers and software have become integral parts of almost every business, the demand for high-quality software is increasing. End users cannot afford system crashes, lost work, or lower productivity. Nor can they tolerate security holes through which intruders can spread viruses, steal data, or shut down Web sites.
- A software defect is any error that, if not removed, could cause a software system to fail to meet its users' needs.
- Software quality is the degree to which a software product meets the needs of its users.
- Software developers are under extreme pressure to reduce the time to market of their products. They are driven by the need to beat the competition in delivering new functionality to users, to begin generating revenue to recover the cost of development, and to show a profit for shareholders.
- The resources and time needed to ensure quality are often cut under the intense pressure to ship a new software product. When forced to choose between adding more user features and doing more testing, many software companies decide in favor of more features.
- Software product liability claims are typically based on strict liability, negligence, breach of warranty, or misrepresentation—sometimes in combination.
- A software development methodology defines the activities in the software development process, defines individual and group responsibilities for accomplishing objectives, recommends specific techniques for accomplishing the objectives, and offers guidelines for managing the quality of the products during the various stages of the development cycle.
- Using an effective development methodology enables a manufacturer to produce high-quality software, forecast project-completion milestones, and reduce the overall cost to develop and support software. An effective development methodology can also help protect software manufacturers from legal liability for defective software in two ways: (1) by reducing the number of software errors that could cause damage and (2) by making negligence more difficult to prove.
- The cost to identify and remove a defect in the early stages of software development can be up to 100 times less than removing a defect in a piece of software that has been distributed to customers.
- Quality assurance (QA) refers to methods within the development cycle designed to guarantee reliable operation of a product. Ideally, these methods are applied at each stage of the development cycle.
- Capability Maturity Model Integration (CMMI)—developed by the Software Engineering Institute at Carnegie Mellon—is a process-improvement approach that defines the

essential elements of effective processes. CMMI defines five levels of software development maturity: initial, managed, defined, quantitatively managed, and optimizing. CMMI identifies the issues that are most critical to software quality and process improvement. Its use can improve an organization's ability to predict and control quality, schedule, costs, and productivity when acquiring, building, or enhancing software systems. CMMI also helps software engineers analyze, predict, and control selected properties of software systems.

- A safety-critical system is one whose failure may cause human injury or death. In the development of safety-critical systems, a key assumption is that safety will *not* automatically result from following an organization's standard software development methodology.
- Safety-critical software must go through a much more rigorous and time-consuming development and testing process than other kinds of software; the appointment of a project safety engineer and the use of a hazard log and risk analysis are common in the development of safety-critical software.
- The International Organization for Standardization (ISO) issued its 9000 series of business management standards in 1988. These standards require organizations to develop formal quality management systems that focus on identifying and meeting the needs, desires, and expectations of their customers.
- The ISO 9001:2008 standard serves as a guide to quality products, services, and management. Approximately 1 million organizations in more than 175 countries have ISO 9001 certification. Many businesses and government agencies specify that a vendor must be ISO 9001 certified to win a contract from them.
- Failure mode and effects analysis (FMEA) is an important technique used to develop ISO 9001-compliant quality systems. FMEA is used to evaluate reliability and determine the effects of system and equipment failures.

Key Terms

black-box testing
breach of warranty
business information system
Capability Maturity Model Integration (CMMI)
CMMI-Development (CMMI-DEV)
contributory negligence
decision support system (DSS)
deliverable
dynamic testing
failure mode
failure mode and effects analysis (FMEA)
hazard log
high-quality software system
integration testing
ISO 9001 family of standards
N-version programming
product liability
quality assurance (QA)
quality management
redundancy
reliability
risk
safety-critical system
software defect
software development methodology
software quality

static testing
strict liability
system safety engineer
system testing
user acceptance testing
warranty
white-box testing

Self-Assessment Questions

The answers to the Self-Assessment Questions can be found in Appendix B.

1. Which of the following is true about a high-quality software system?
   a. It is more difficult to learn and use.
   b. It meets its users' needs.
   c. It operates more slowly and deliberately.
   d. It operates in an unreliable manner.
2. Software ______________ is the degree to which a software product meets the needs of its users.
3. Which of the following is a major cause of poor software quality?
   a. Many developers do not know how to design quality into software or do not take the time to do so.
   b. Programmers make mistakes in turning design specifications into lines of code.
   c. Software developers are under extreme pressure to reduce the time to market of their products.
   d. All of the above are major causes of poor software quality.
4. A decision support system might be used to do which of tne following?
   a. process large numbers of business transactions
   b. assist managers in developing accurate forecasts
   c. control manufacturing processes
   d. perform all of the above
5. The liability of manufacturers, sellers, lessors, and others for injuries caused by defective products is commonly referred to as ______________.
6. A standard, proven work process for the development of high-quality software is called a(n) ______________.
7. The cost to identify and remove a defect in an early stage of software development is typically about the same as the cost of removing a defect in an operating piece of software after it has been distributed to many customers. True or False?
8. A software ______________ is any error that if not removed could cause a software system to fail to meet its users' needs.
9. Methods within the development cycle designed to guarantee reliable operation of the product are known as ______________.

10. Which of the following is a form of software testing that involves viewing a software unit as a device that has expected input and output behaviors but whose internal workings are known?
    a. dynamic testing
    b. white-box testing
    c. integration testing
    d. black-box testing
11. Which of the following is an approach that defines the essential elements of an effective process and outlines a system for continuously improving software development?
    a. CMMI-DEV
    b. FMEA
    c. ISO-9000
    d. DOD-178B
12. One of the most important and difficult areas of safety-critical system design is the system-human interface. True or False?
13. The provision of multiple interchangeable components to perform a single function to cope with failures and errors is called:
    a. risk
    b. redundancy
    c. reliability
    d. availability
14. A reliability evaluation technique that can determine the effect of system and equipment failures is ______________.
15. When discussing system performance, the terms reliability and safety mean the same. True or False?
16. In a lawsuit alleging ______________, responsibility is limited to harmful defects that could have been detected and corrected through "reasonable" software development practices.

Discussion Questions

1. Identify the three criteria you consider to be most important in determining whether or not a system is a quality system. Briefly discuss your rationale for selecting these criteria.
2. Briefly describe and give an example of a business information system, a decision support system, and a control system.
3. Define and briefly discuss the difference between white box testing and black box testing.
4. Explain why the cost to identify and remove a defect in the early stages of software development might be 100 times less than the cost of removing a defect in software that

has been distributed to customers. What are the implications for a software development organization?

5. Explain the difference between strict liability and negligence.
6. Identify and briefly discuss two ways that the use of an effective software development methodology can protect software manufacturers from legal liability for defective software.
7. Your company is considering using N-version programming with three software development firms and three hardware devices for the navigation system of a guided missile. Briefly describe what this means, and outline several advantages and disadvantages of this approach.
8. Why is the system-human interface one of the most important but difficult areas of safety-critical systems? Do a search on the Internet and find three good sources of information relating to how to design an effective system-human interface.
9. What is the difference between system reliability and system safety? Give an example of a system that operates reliably but not safely.
10. Identify and briefly discuss the implications to a project team of classifying a piece of software as safety critical.
11. Your organization develops accounting software for use by individuals to budget and forecast their expenses and pay their bills while keeping track of the amount of money in their savings and checking accounts. Develop a strong argument for the management of your firm as to why the firm must conduct an assessment of its current software development practices.
12. Discuss why an organization might elect to use a separate, independent team for quality testing rather than the group of people who originally developed the software.

What Would You Do?

Use the five-step decision-making process discussed in Chapter 1 to analyze the following situations and recommend a course of action.

1. Read the fictional Killer Robot case at the Web site for the Online Ethics Center for Engineering at *www.onlineethics.com/CMS/computers/compcases/killerrobot.aspx*. The case begins with the manslaughter indictment of a programmer for writing faulty code that resulted in the death of a robot operator. Slowly, over the course of many articles, you are introduced to several factors within the corporation that contributed to the accident. After reading the case, answer the following questions:
   a. Responsibility for an accident is rarely defined clearly and is often difficult to trace to one or two people or causes. In this fictitious case, it is clear that a large number of people share responsibility for the accident. Identify all the people you think were at least partially responsible for the death of Bart Matthews, and explain why you think so.
   b. Imagine that you are the leader of a task force assigned to correct the problems uncovered by this accident. Develop a list of the six most significant actions to take to avoid future problems.

2. Your manager is leading a project to develop new software that is essential to the success of the midsized manufacturing firm where you work. The firm has decided to hire outside contractors to execute the project. One candidate firm boasts that its software development practices are at level 4 of CMMI. Another firm claims that all its software development practices are ISO 9001 compliant. Your manager has come to you and asked for your opinion on how much weight should be given to these certifications when deciding which firm to use. What would you say?
3. You are a programmer for a firm that develops a popular tax preparation software package designed to help individuals prepare their federal tax returns. In the course of testing some small changes that were made to the software, you detect an error in the software that results in roughly a 5 percent underestimation of the amount owed—both for those who indicated that they were single and for those who indicated that they were married but filing separate tax returns. It is now late March, and it is likely that well over 100,000 users who submitted their returns using your firm's software will be affected by this error. What do you do?

4. You are the project manager in charge of developing the latest release of your software firm's flagship product. The product release date is just two weeks away, and enthusiasm for the product is extremely high among your customers. Stock market analysts are forecasting sales of more than $25 million per month. If so, earnings per share will increase by nearly 50 percent. There is just one problem: two key features promised to customers in this release have several bugs that would severely limit the software's usefulness. You estimate that at least six weeks are needed to find and fix the problems. In addition, even more time is required to find and fix 15 additional, less severe bugs just uncovered by the QA team. What would you recommend to management?
5. You developed a spreadsheet program that helps you perform your role of inventory control manager at a small retail sports shoe store. The software uses historical sales data to calculate expected weekly sales for each of about 250 shoes carried by the store. Based on that forecast, you order the appropriate shoes from the various manufacturers. Your store is one of four shoe stores owned by the same person. You sent a copy of the spreadsheet to each of the people responsible for inventory control at the other three retail stores, and they are all now using your software to help them do their jobs. You have started getting complaints that the software is not entirely accurate, and you notice that your own estimates are no longer as accurate as they used to be. What would you do?
6. You have been assigned to manage software that controls the shutdown of the new chemical reactors to be installed at a manufacturing plant. Your manager insists the software is not safety critical. The software senses temperatures and pressures within a 50,000-gallon stainless steel vat and dumps in chemical retardants to slow down the reaction if it gets out of control. In the worst possible scenario, failure to stop a runaway reaction would result in a large explosion that would send fragments of the vat flying and spray caustic liquid in all directions.

   Your manager points out that the stainless steel vat is surrounded by two sets of protective concrete walls and that the reactor's human operators can intervene in case of a software failure. He feels that these measures would protect the plant employees and the surrounding neighborhood if the shutdown software failed. Besides, he argues, the plant is already more than a year behind its scheduled start-up date. He cannot afford the additional time required to develop the software if it is classified as safety critical. How would you work with your manager and other appropriate resources to decide whether the software is safety critical?

7. You are a senior software development consultant with a major consulting firm. You have been asked to conduct a follow-up assessment of the software development process for ABCXYZ Corporation, a company for which you had performed an initial assessment using CMMI two years prior. At the initial assessment, you determined the company's level of maturity to be level 2. Since that assessment, the organization has spent a lot of time and effort following your recommendations to raise its level of process maturity. The organization appointed a senior member of its IT staff to be a process management guru and paid him $150,000 per year to lead the improvement effort. This senior member adopted a methodology for standard software development and required all project managers to go through a one-week training course at a total cost of more than $2 million.

   Unfortunately, these efforts did not significantly improve process maturity because senior management failed to hold project managers accountable for actually using the standard development methodology in their projects. Too many project managers convinced senior management that the new methodology was not necessary for their particular project and would just slow things down. You are concerned that when senior management learns that no real progress has been made, they will refuse to accept partial blame for the failure and instead drop all attempts at further improvement. You are also likely to lose your contract with the firm. What would you do?

8. You are the CEO for a small, struggling software firm that produces educational software for high school students. Your latest software is designed to help students improve their SAT and ACT scores. To prove the value of your software, a group of 50 students who had taken the ACT test were retested after using your software for just two weeks. Unfortunately, there was no dramatic increase in their scores. A statistician you hired to ensure objectivity in measuring the results claimed that the variation in test scores was statistically insignificant. You had been counting on touting the results in the promotion of your new software.

   A small core group of educators and systems analysts will need at least six months to start again from scratch to design a viable product. Programming and testing could take another six months. Another option would be to go ahead and release the current version of the product and then, when the new product is ready, announce it as a new release. This would generate the cash flow necessary to keep your company afloat and save the jobs of 10 or more of your 15 employees. Given this information about your company's product, what would you do?

Cases

1. InterSystems Earns ISO 9001-2008 Certification

InterSystems is a privately held software development firm with recent sales revenue of $446 million. The company is headquartered in Cambridge, Massachusetts, with offices in 25 countries worldwide.[40] Recently, InterSystems became ISO 9001:2008 certified for all processes related to product and service creation in connection with two of its primary products: Caché and Ensemble.[41] By meeting these requirements, InterSystems has proven that it has in place systems and processes necessary to ensure that its products and services are delivered in a controlled and repeatable manner. ISO 9001-2008 certification is proof of an organization's commitment to quality management and continuous improvement.

The Caché product is a high-performance/high-reliability database management system. The software comes bundled with an application development environment that assists programmers in the rapid development of software applications. Caché is used extensively by organizations in clinical healthcare applications to develop systems that capture, organize, and analyze healthcare records in ways that lead to better patient experiences and improved healthcare outcomes.[42]

The Johns Hopkins Cancer Center, nationally recognized as one of the leading cancer centers in the United States, is a major InterSystems customer. The hospital implemented an advanced, multifunctional oncology clinical information system based on Caché. The system records all interactions among patients, caregivers, providers, and administrators from the time they register to enter the facility until they leave and are billed. During a typical visit to the center, patients have multiple appoinments with various care providers and undergo various tests and treatments. Patients are issued a bar-coded ID that is scanned at strategic locations as they move through the hospital—allowing personnel to track what appointments remain and where the patient is at any time. Key data associated with all tests, treatments, and patient results is captured so that care providers can review treatment approaches used in the past to help decide the best treatment process for new patients.[43]

Discussion Questions

1. A mission-critical system is one whose failure will result in an organization being unable to continue business operations. A safety-critical system is one whose failure will result in human injury or loss of life. Is the John Hopkins system described above mission critical or safety critical? Why? Can you give an example of a safety-critical system that is not mission critical?
2. Caché and its associated application tools constitute a system that is used to build a wide variety of information systems for customers around the world. Do you think that the Caché software and tools should be considered a safety-critical system and undergo the rigorous development process associated with such systems? If so, what would be the implications for InterSystems and its customers in terms of costs and frequency of software modifications and updates? Would this put InterSystems at a competitive disadvantage to other software development companies?
3. Should every organization that builds safety-critical systems be required to have all its system development processes and tools ISO-9000: 2008 certified? Why or why not?

2. Apple Guidelines for App Approval

Apple's App Store has been a huge success ever since it was launched in 2008. As of April 2013, the App Store offered more than 500,000 applications available for sale to owners of Apple iPhone, iPad, and iPod devices—with more than 4 billion downloads in the first quarter of 2013.[44]

Before software applications can be sold through the App Store, they must go through a review process. Apple has been accused by some of using clandestine and capricious rules to reject some programs—thus, blocking them from reaching the very large and growing market of iPhone, iPad, and iPod Touch users. One application developer complained: "If you submit an app, you have no idea what's going to happen. You have no idea when it's going to be approved or if it's going to be approved."[45] The developers of an app called "South Park" complained that their app was rejected because the content was deemed "potentially offensive," even though episodes of the award-winning animated sitcom are available at the Apple iTunes Store.[46] In September 2010, after more than two years of complaints, Apple finally provided application developers the guidelines it uses to review software.

Most guidelines seem to be aimed at ensuring that Apple users can only access high-quality and noncontroversial apps from its App Store. Some of the Apple guidelines are clear and their rationale is easy to understand, such as "apps that rapidly drain the device's battery or generate excessive heat will be rejected." However, other guidelines are unclear and highly subjective, such as "We will reject apps for any content or behavior that we believe is over the line. What line, you ask? Well, as a Supreme Court Justice once said, 'I'll know it when I see it.' And we think that you will also know it when you cross it."[47] ("I know it when I see it" was the phrase used by U.S. Supreme Court Justice Potter Stewart to describe his ability to recognize rather than to provide a precise definition of hard core pornography in his opinion in the case *Jacobellis v. Ohio* in 1964.)

The Electronic Frontier Foundation believes that while the guidelines are helpful, in some cases Apple is defining the content of third-party software and placing limits on what is available to customers of Apple's App Store.[48]

By way of comparison, Google places few restrictions on developers of software for its competing Android Marketplace. However, there have been many low-quality applications offered to Android Marketplace customers, including some that include malware. Indeed by early 2011, Google had pulled 21 Android applications from its Android Marketplace because, once downloaded, the applications not only stole users' information and device data, but also created a backdoor for even more harmful attacks.[49] Apple's decision to finally share its applications guidelines may have been an attempt to combat the rapidly increasing popularity of the Android.[50] It may also have been a response to a U.S. Federal Trade Commission investigation of a complaint from Adobe concerning Apple's banning of the Flash software from devices that run Apple's iOS operating system. (Adobe® Flash® Player is a browser-based application that runs on many computer hardware/operating system combinations and supports the viewing of so called "rich, expressive applications," content, and videos across screens and browsers.)[51]

Discussion Questions

1. Should Apple conduct extensive screening of apps before they are allowed to be sold on the App Store? Why or why not?
2. Do research to determine the current status of the FCC investigation of Apple for banning use of the Adobe Flash software on devices that use the iOS operating system.
3. What do you think of Apple's guideline that says it will reject an app for any content or behavior that they believe is "over the line"? Could such a statement be construed as a violation of the developer's freedom of speech? Why or why not?

3. Software Errors Lead to Death

Medical linear accelerators have long been a critical piece of medical equipment in the fight against cancer. Linear accelerators deliver radiation therapy to cancer patients by accelerating electrons to create high-energy beams, which can kill cancer tumors without impacting surrounding healthy tissue. Tumors close to the skin can be treated with the accelerated electrons; however, for tumors that are more deeply embedded, the electron beam is converted into an X-ray photon beam, which is diffused using a beam spreader plate.

The Canadian firm Atomic Energy of Canada Limited (AECL) and a French company named CGR collaborated to build two models of medical linear accelerators. One model, the Therac-6, was capable of producing only X-rays that could be used to kill tumors close to the skin. A later model, the Therac-20, was capable of producing both X-ray photons and electrons and thus could kill both shallow and deeply embedded tumors. Computer software was used to simplify the operation of the equipment but not to control and monitor its operation. Instead, industry standard hardware safety features were built into both models.[52]

After the business relationship between the two firms failed, AECL went on to build the Therac-25 based on a new design concept. Unlike the Therac-6 and Therac-20, which operated without significant computer controls, computer software was used to both control and monitor the Therac-25 accelerator.[53]

The software for the Therac-25 was based on modified code from the Therac-6. The software monitored the machine, accepted technician input for specific patient treatment, initialized the machine to administer the defined treatment, and controlled the machine to execute the defined treatment. The machine was enclosed in the patient treatment room to prevent radiation exposure to the technicians. Audio and visual equipment allowed the patient to communicate with the technicians.[54]

A total of 11 Therac-25 machines were installed in the United States and Canada. Over a 19-month period from June 1985 to January 1987, six serious incidents involving the use of the device occurred. In each of the incidents, the patient received an overdose of radiation. Four of the patients died from the overdose, and another eventually had to have both breasts removed and lost use of her right arm as a result of the overdose. A final patient received burns and was only able to fully recover several years after the incident.[55]

Following each incident, AECL was contacted and asked to investigate the situation. However, AECL at first refused to believe that its machine could have been responsible for an overdose. Indeed, following the third incident, AECL responded, "After careful consideration, we are of the opinion that this damage could not have been produced by any malfunction of the Therac-25 or by any operator error." AECL made some minor changes to the equipment, but because the company did not address the root cause of the problem, additional incidents occurred.[56]

Finally, a physicist at a hospital where two incidents occurred was able to re-create the malfunction and show that the problem was due to a defect in the machine and its software. A failure occurred when a specific sequence of keystrokes was entered by the operator. Because this sequence of keystrokes was nonstandard, the problem rarely occurred and went undetected for a long time. Entry of this combination of keystrokes within a period of eight seconds did not allow time for the beam spreader plate to be rotated into place. The software did not recognize the error, and the patient was then hit with a high-powered electron beam roughly 100 times the intended dose of radiation.[57]

In early 1987, the Food and Drug Administration (FDA) and Health Canada (the Canadian counterpart to the FDA) insisted that all Therac-25 units be shut down. Within the next six months, AECL implemented numerous code changes, installed independent hardware safety locks, and implemented other changes to correct the problem.[58] After these changes, the Therac-25 device continued to be safely used for many years. However, at least three lawsuits were filed against AECL and the hospitals involved in the earlier incidents. The lawsuits were settled out of court, and the results were never revealed.[59]

Discussion Questions

1. What additional measures must be taken in the development of software that, if it fails, can cause loss of human life?
2. What can organizations do to reduce the negative consequences of software development problems in the production of their products and the operation of their business processes and facilities?

End Notes

1 "Regulation NMS," NASDAQ OMX, www.nasdaqtrader.com/Trader.aspx?id=RegNMS (accessed March 23, 2013).

2 Columbia Business School, "Press Release: High-Frequency Trading: Is It Good or Bad for Markets?", *Yahoo! Finance*, March 20, 2013, http://finance.yahoo.com/news/high-frequency-trading-good-bad-130000973.html.

3 Ben Rooney, "Trading Program Sparked May 'Flash Crash,'" *CNN Money*, October 1, 2010, http://money.cnn.com/2010/10/01/markets/SEC_CFTC_flash_crash/index.htm.

4 Ben Rooney, "Trading Program Sparked May 'Flash Crash,'" *CNN Money*, October 1, 2010, http://money.cnn.com/2010/10/01/markets/SEC_CFTC_flash_crash/index.htm.

5 BATS Global Markets, "Overview: About Us," www.batsglobalmarkets.com (accessed March 20, 2013).

6 Melanie Rodier, "Flash Crash at BATS Renews Market Concerns," *Wall Street & Technology*, March 23, 2012, www.wallstreetandtech.com/electronic-trading/flash-crash-at-bats-renews-market-concer/232700195.

7 D.M. Levine, "Apple Has Mini Flash Crash on BATS," *Huffington Post*, March 23, 2012, www.huffingtonpost.com/2012/03/23/flash-crash-apple-stock-bats_n_1375496.html.

8 Phil Albinus, "BATS Flash Crash: Here's What Happened," *Advanced Trading*, March 26, 2012, www.advancedtrading.com/exchanges/bats-flash-crash-heres-what-happened/232700223.

9 Pallavi Gogoi, "Knight Capital Blames Software for Computer Trading Glitch," *USA Today*, August 2, 2012.

10 Pallavi Gogoi, "Knight Capital Blames Software for Computer Trading Glitch," *USA Today*, August 2, 2012.

11 Pallavi Gogoi, "Knight Capital Blames Software for Computer Trading Glitch," *USA Today*, August 2, 2012.

12 Ivy Schmerken, "Another Technical Issue from BATS Rattles Confidence," *Advanced Trading*, January 10, 2013, www.advancedtrading.com/exchanges/another-technical-issue-from-bats-rattle/240146030.

13 Greg Bensinger, "Software Glitch Mars Nokia's US Re-Entry," *Wall Street Journal* (blog), April 11, 2012, http://blogs.wsj.com/digits/2012/04/11/software-glitch-mars-nokias-us-re-entry-with-att.

14 "IRS Software Glitch Delays Some Tax Refunds," Reuters, March 3, 2012, www.rawstory.com/rs/2012/03/03/irs-software-glitch-delays-some-tax-refunds.

15 "Software Glitch Could Strand Chevy Volt Drivers," *Fox News*, October 23, 2012, www.foxnews.com/leisure/2012/10/23/software-glitch-could-strand-chevy-volt-drivers.

16 Lauren Walsh, "Software Error Prevents Many Georgia Stores from Selling Powerball Tickets," WAGT 26, November 25, 2012, www2.nbc26.tv/news/2012/nov/25/software-error-prevents-many-georgia-stores-sellin-ar-5045240.

17 Chelsea Bannach, "WSU Software Glitch Stymies Students," *Spokesman-Review*, August 22, 2012, www.spokesman.com/stories/2012/aug/22/wsu-software-glitch-stymies-students.

18 Katherine Noyes, "Actually, Open Source Code Is Better: Report," *PC World*, February 23, 2012, www.pcworld.com/article/250543/actually_open_source_code_is_better_report.html.

19 Katherine Noyes, "Actually, Open Source Code Is Better: Report," *PC World*, February 23, 2012, www.pcworld.com/article/250543/actually_open_source_code_is_better_report.html.

20 Klaus Enzenhofer, "Mobile App Performance – How to Ensure High Quality Experiences," *Testing Experience*, September 19, 2012, www.testingexperience.com/testingexperience19_09_12.pdf.

21 Cem Kaner, "Quality Cost Analysis: Benefits and Risks," *Software Quality Assurance* 3, no. 1 (1996), www.kaner.com/pdfs/Quality_Cost_Analysis.pdf (accessed March 23, 2013).

22 Paul Bibby, "Qantas Exposed to Compo Claims," *WA Today*, October 9, 2008, www.watoday.com.au/national/qantas-exposed-to-compo-claims-20081009-4x6t.html.

23 Martin Samson, *"M. A. Mortenson Co. v. Timberline Software Co. et al."* *Internet Library of Law and Court Decisions*, www.internetlibrary.com/cases/lib_case206.cfm (accessed March 23, 2013).

24 Barry W. Boehm, "Improving Software Productivity," *IEEE Computer* 20, no. 8 (1987): 43–58.

25 Capers Jones, *"Software Quality in 2002: A Survey of the State of the Art,"* Software Productivity Research, Inc., November 2002.

26 Dennis R. Goldenson and Diane L. Gibson, "Demonstrating the Impact and Benefits of CMMI: An Update and Preliminary Results," October 2003, www.sei.cmu.edu/reports/03sr009.pdf.

27 Northrup Grumman, "Press Release: Northrop Grumman's Rolling Meadows Campus Achieves CMMI(R) Maturity Level 5 Rating," *Globe Newswire*, December 21, 2012, www.irconnect.com/noc/press/pages/news_releases.html?d=10016400.