Software Quality and Management:

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..

Causes of Poor Software Quality:

- **Lack of Design Quality:** Many developers don't know how to design quality into software from the start.
- **Human Error:** Even experienced developers introduce defects. Studies indicate an average of defects per lines of code written. every 7 to 10 lines of code
- **Time-to-Market Pressure:** Companies prioritize speed to market over thorough testing to beat competitors, recover costs, and meet financial targets.
- **Feature Over Testing:** Companies often choose to add features over comprehensive testing, relying on future patches.
- **Ethical Dilemmas:** Balancing the cost and effort of quality assurance against customer expectations is a major ethical issue.
- **Changing Operating Conditions:** Software that has been reliable can fail when operating conditions change unexpectedly.

Prevalence of Defects:

- Commercial software contains many defects.
- Open-source code has been shown to have fewer defects on average than commercial code.
- Due to defects, many organizations avoid first releases of software.

Customer Impact:

Customers are stakeholders who benefit from new features but bear the burden of undiscovered defects.

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.

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. 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.

Software as a Business Necessity:

Many companies are now heavily reliant on software, making software quality, usability, and development speed critical competitive factors. Software problems can cause delays, increase costs, and damage product quality, potentially leading to business failure.

Ethical Considerations:

- **Balancing Profit and Quality:** Executives face ethical dilemmas regarding the investment of time and resources in software quality assurance. Short-term profit goals may conflict with long-term quality and customer satisfaction.
- **Product Liability:** Executives must consider the potential for damage caused by defective software and their legal exposure. While lethal software defects are rare, product liability concerns are significant.

Customer Impact:

Poor quality software negatively affects customers, and can lead to customer dissatisfaction, and loss of business.

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.

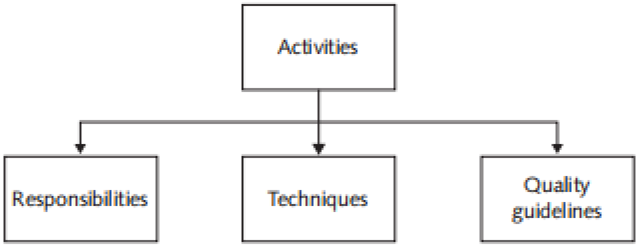

FIGURE 7-1 Software development methodology
Source Line: Course Technology/Cengage Learning.

Importance of Early Defect Detection:

Identifying and fixing defects early in the development process is significantly cheaper than addressing them later. Studies show that the cost of fixing a defect during the requirements definition stage can be up to 100 times less than fixing it after the software is distributed to customers. Late-stage defects impact more people, requiring more communication, fixes, and potential retraining.

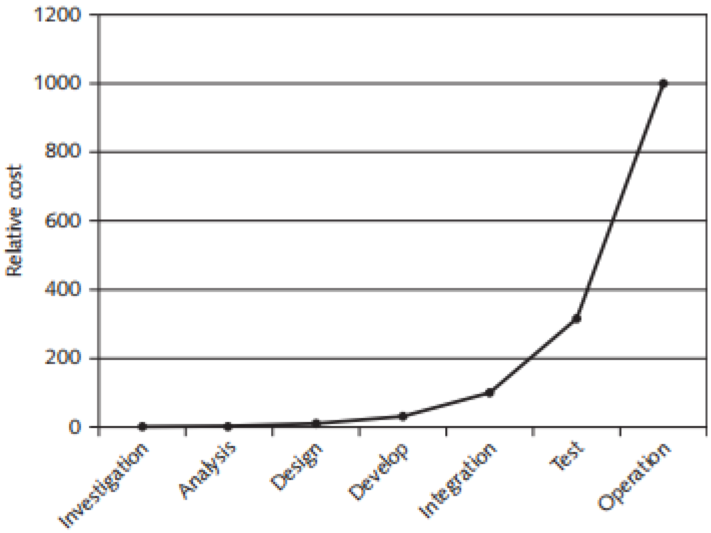

FIGURE 7-2 The cost of removing software defects

Cost-Saving and Quality Improvement:

Early defect detection is not only cost-effective but also the most efficient way to improve software quality. It reduces the overall effort required to deliver a reliable product.

Product Liability and Legal Protection:

Defective software can lead to product liability lawsuits. Using an effective methodology can:

- Reduce the number of errors.
- Make it harder to prove negligence, as the development process follows accepted standards.

However, even a successful defense can be expensive, highlighting the importance of careful and consistent software development to minimize liability exposure.

Dynamic Software Testing

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 c**onsider 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

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 (CMMI):

CMMI, developed by the Software Engineering Institute at Carnegie Mellon, is a **process-improvement framework**. It defines the essential elements of effective processes and can be used to evaluate and improve virtually any process. CMMI-DEV is specifically designed for software development.

CMMI Maturity Levels:

CMMI defines five maturity levels (see Table 7-1):

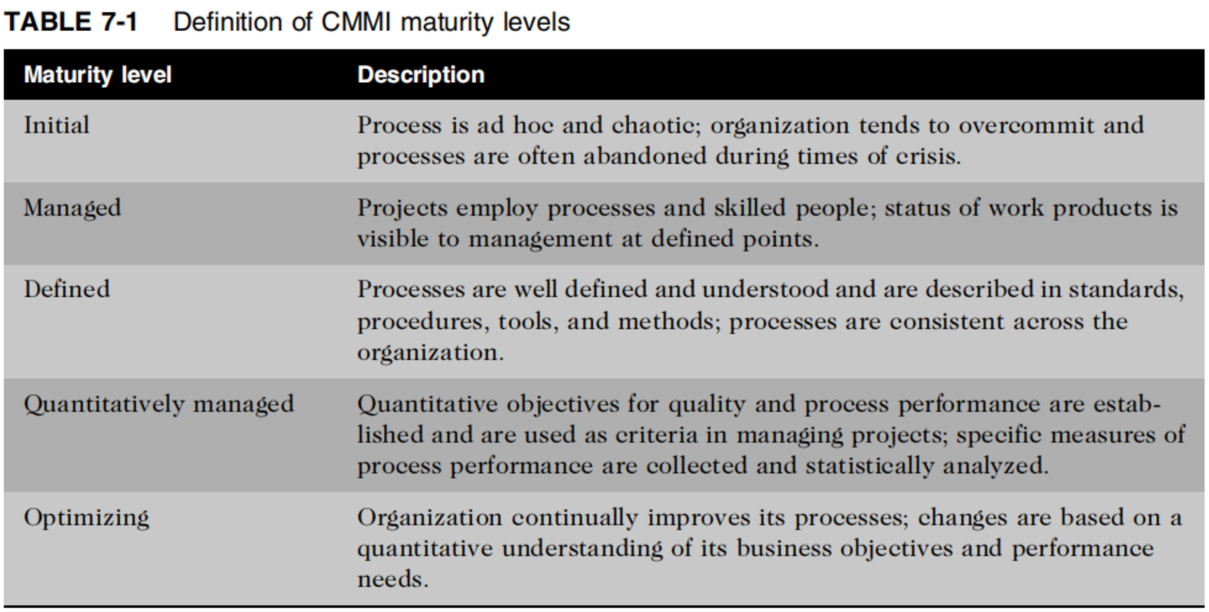

TABLE 7-1 Definition of CMMI maturity levels

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.

Quantitatively Managed: Quantitative objectives are set, and process performance is measured and analyzed.

Optimizing: Processes are continually improved based on quantitative understanding.

.

Benefits of CMMI:

CMMI helps organizations improve their performance by:

- Defining a set of practices for process improvement.
- Identifying the organization's current maturity level.
- Enabling tracking and evaluation of progress.

Studies have shown that implementing CMMI can lead to:

- Reduced defect fixing costs.
- Lower software costs.
- Increased productivity.
- Improved project schedule adherence.

CMMI-DEV:

CMMI-DEV is a set of guidelines for 22 process areas related to systems development. It emphasizes that organizations that excel in these areas will have outstanding software development processes.

Implementation of CMMI-DEV:

Organizations adopting CMMI-DEV conduct assessments to determine their current maturity level and identify areas for improvement. They develop and execute action plans to move to higher maturity levels.

CMMI as a Benchmark:

CMMI-DEV can be used to compare organizations, particularly in government contracting, where bidders may be required to have a certain CMMI level.

Achieving Maturity Level 5:

Reaching Maturity Level 5 signifies that an organization can statistically evaluate its processes, leading to better control, continuous improvement, and the ability to deliver high-quality software on time and within budget. The example of Northrop Grumman highlights the benefits of achieving Level 5.