SYSTEMS AUDITING

(DMCA305)

(MCA)

ACHARYA NAGARJUNA UNIVERSITY

CENTRE FOR DISTANCE EDUCATION

NAGARJUNA NAGAR,

GUNTUR

ANDHRA PRADESH

CONTENTS

UNIT – I

1. Overview of Information Systems Auditing

Structure

Objectives

In this lesson you will learn about:

- Need for control and audit of computers
- Information systems auditing
- Effects of computers on internal controls
- Effects of computers on auditing
- Foundations of information systems auditing

1.1 Introduction

Whereas 50 years ago we fulfilled most of our data processing needs manually, today computers perform much of the data processing required in both the private and public sectors of our economies. As a result, the need to maintain the integrity of data processed by computers now seems to pervade our lives. Many people fear that our substantially increased data-processing capabilities are not well controlled. The media make much of computer abuse. We have concerns about the privacy of data we exchange with organizations such as the tax department, medical authorities, and credit granting institutions. All of us have probably suffered the frustrations of trying to get an organization to update its computer-maintained name and address file.

Uncontrolled use of computers can have a widespread impact on a society. For example, inaccurate information causes misallocation of resources within the economy, and fraud can be perpetrated because of inadequate system controls. Unfortunately, those who suffer most often are those who can least afford to suffer for example, small shareholders and low-income earners. Perhaps more subtle is the growing distrust of institutions that gather and process large volumes of data. Because computers now seem to be ubiquitous, many people have a sense of lost individuality: The "big brother" of George Orwell's 1984 is upon us.

1.2 Need for control and audit of computers

Computers are used extensively to process data and to provide information for decision making. Initially, they were available only to large organizations that could afford their high purchase and operation costs. The advent of minicomputers and the rapid decrease in the cost of computer technology then enabled medium-sized organizations to take advantage of computers for their data processing. Nowadays, the widespread availability of powerful microcomputers and their associated packaged software has resulted in the extensive use of computers in the workplace and at home. Given the intensely competitive marketplace for computer hardware and software technology, the rapid diffusion of computers in our economies will continue.

Because computers play such a large part in assisting us to process data and to make decisions, it is important that their use be controlled. Figure 1-1 shows seven major reasons for establishing a function to examine controls over computer-based data processing. In the following subsections, we examine these reasons in more detail.

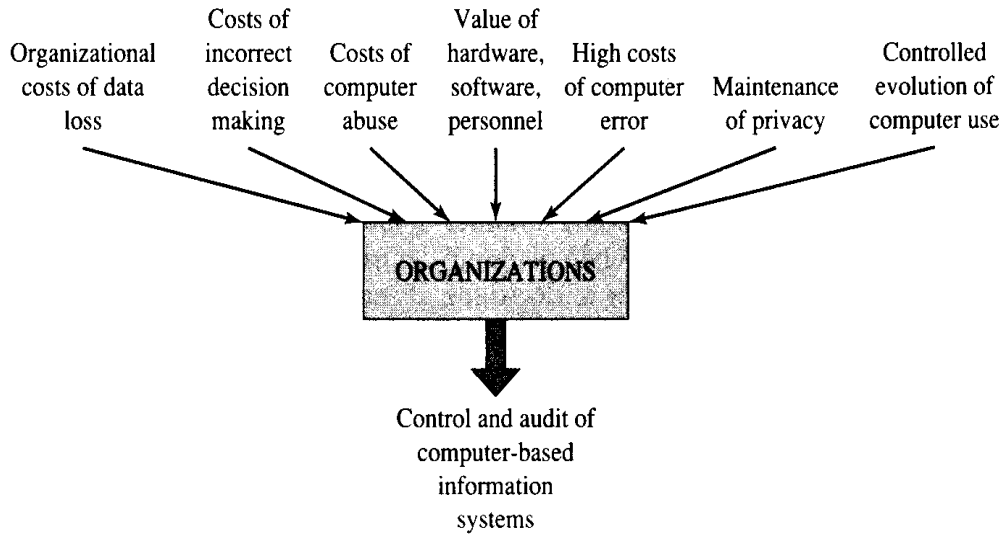


FIGURE 1-1 Factors influencing an organization toward control and audit of computers.

1.2.1 Organizational Costs of Data Loss

Data make up a critical resource necessary for an organization's continuing operations. In this regard, Everest (1985) proposes that data provides the organization with an image of itself, its environment, its history, and its future. If this image is accurate, the organization increases its abilities to adapt and survive in a changing environment. If this image is inaccurate or lost, the organization can incur substantial losses.

For example, consider a large department store whose accounts receivable file has been destroyed. Unless its customers are honest and also remember what they have purchased from the store, the firm can suffer a major loss in cash receipts through customers failing to pay their debts. The department store's long-run survival could be affected. Consider, also, a department store that loses its accounts payable file. Most likely it will be unable to pay its debts on time. As a result, it could suffer a loss of credit rating as well as any discounts available for early payment. If it contacts creditors requesting their assistance, the department store has to rely on the honesty of its creditors in notifying it of the amounts it owes. Furthermore, creditors might now begin to question the competence of the department store's management. As a result, they might be unwilling to extend credit to the department store in the future.

Such losses can occur when existing controls over computers are lax. For example, management might not provide adequate backup for computer files. Thus, the loss of a file through computer program error, sabotage, or natural disaster means the file cannot be recovered, and the organization's continuing operations are thereby impaired.

1.2.2 Incorrect Decision Making

Making high-quality decisions depends in part on the quality of the data and the quality of the decision rules that exist within computer-based information systems. Let's consider the significance of both in turn.

The importance of accurate data in a computer system depends on the types of decisions made by persons having some interest in an organization. For example, if managers are making strategic planning decisions, they will probably tolerate some errors in the data, given the long-run nature of strategic planning decisions and the inherent uncertainty surrounding these types of decisions. If managers are making management control and operational control decisions, however, they will probably require highly accurate data. These types of decisions involve detection, investigation, and correction of out-of-control processes. Thus, inaccurate data can cause costly, unnecessary investigations to be undertaken or out-of-control processes to remain undetected.

Besides management, incorrect data can also have an impact on other parties who have an interest in an organization. For example, shareholders might make poor investment decisions if they are provided with inaccurate financial information. Similarly, governments, labor, and lobby groups might make poor decisions if they are provided with inaccurate or incomplete data about an organization. They might begin to make demands on the organization (e.g., for control of greenhouse emissions or higher wages) that the organization cannot sustain.

The importance of having accurate decision rules in a computer system also depends on the types of decisions made by persons having some interest in an organization. In some cases, an incorrect decision rule can have minor consequences. For example, a small, inconsequential error can occur in the

calculation of depreciation on a low-value asset. In other cases, however, the consequences can be significant. For example, if the algorithm that determines the interest rate to be paid to customers of a bank is incorrect, the bank might make substantial overpayments to its customers. It might not be able to recover these monies without substantial losses of goodwill. Similarly, if a decision rule in an expert system that supports medical diagnosis is incorrect, doctors could prescribe inappropriate treatments for patients, some of which could be fatal.

1.2.3 Costs of Computer Abuse

The major stimulus for development of the information systems audit function within organizations often seems to have been computer abuse. Parker defines computer abuse to be "any incident associated with computer technology in which a victim suffered or could have suffered loss and a perpetrator by intention made or could have made gain."

Some major types of computer abuse that an organization might encounter include the following:

Type of Abuse	Explanation
Hacking	A person gains unauthorized access to a computer system to read, modify, or delete programs or data or to disrupt services.
Viruses	Viruses are programs that attach themselves to executable files, system areas on diskettes, or data files that contain macros to cause disruption to computer operations or damage to data and programs. They are designed to achieve two objectives: to replicate themselves and to deliver a payload that causes disruption of some kind.
Illegal physical access	A person gains unauthorized physical access to computer facilities (e.g., they gain illegal entry to a computer room or a terminal). As a result, they are able to cause physical damage to hardware or make unauthorized copies of programs or data.
Abuse of privileges	A person uses the privileges they have been assigned for unauthorized purposes (eg., they make unauthorized copies of sensitive data they are permitted to access).

Computer abuse can lead to the following types of consequences:

Consequences of Abuse	Explanation
Destruction of assets	Hardware, software, data, facilities, documentation, or supplies can be destroyed.
Theft of assets	Hardware, software, data, documentation, or supplies can he illegally removed.
Modification of assets	Hardware, software, data, or documentation can he modified in an unauthorized way.
Privacy violations	The privacy of data pertaining to a person or an organization can be compromised.
Disruption of operations	The day-to-day operations of the information systems function can cease temporarily.
Unauthorized use of assets	Hardware, software, data, facilities, documentation, or supplies are used for unauthorized purposes (e.g., computer time is used for private consulting purposes).
Physical harm to personnel	Personnel can suffer physical harm.

Computer abuse usually is a less serious problem for organizations than errors and omissions in computer systems or the effects of natural and human made disasters (such as floods and fires). Nonetheless, organizations now appear to be encountering a high incidence of computer abuse. Furthermore, the average losses incurred from computer abuse seem to be substantially higher than those incurred from conventional fraud.

In addition, the number and types of threats that lead to computer abuse also seem to be increasing. For example, as of November 1996, Nachenberg reports that more than 10,000 DOS-based computer viruses had bee written. Moreover, more complex, more lethal viruses continue to appear. Similarly, the rapid growth of the Internet has exposed organizations with inadequate security to many threats from outside hostile parties that previously would not have affected them.

Unfortunately, surveys continue to indicate that a large number of organizations are not well prepared to deal with computer abuse. For example Benbow reports that 80 percent of computer abuse cases investigate in his research were committed by internal employees, but only 20 percent (the organizations he studied performed security reviews of

potential employs. Several surveys also have reported that the chances of many organizations surviving a major incident of computer abuse are poor. A substantial number of organizations could operate for only a few hours without their computer systems, and many would be out of business within a few days.

1.2.4 Value of Computer Hardware, Software, and Personnel

In addition to data, computer hardware, software, and personnel are critical organizational resources. Some organizations have multimillion dollar investments in hardware. Even with adequate insurance, the intentional or unintentional loss of hardware can cause considerable disruption. Similarly, software often constitutes a considerable investment of an organization's resources. If the software is corrupted or destroyed, the organization might be unable to continue operations if it cannot recover the software promptly. If the software is stolen, confidential information could be disclosed to competitors; or, if the software is a proprietary package, lost revenues or lawsuits could arise. Finally, personnel are always a valuable resource, particularly in light of an ongoing scarcity of well-trained computer professionals in many countries.

1.2.5 High Costs of Computer Error

Computers now automatically perform many critical functions within our society. For example, they monitor the condition of patients during surgery, direct the flight of a missile, control a nuclear reactor, and steer a ship on its course. Consequently, the costs of a computer error in terms of loss of life, deprivation of liberty, or damage to the environment can be high. For example, data errors in a computer system used to control flight paths resulted in the death of 257 people when an airplane crashed into a mountain in Antarctica; a person was jailed incorrectly for five months because of erroneous data contained in a computer system.

The costs of computer error in financial terms can also be high. An error in an Australian government computer system resulted in a $126 million overpayment of pharmaceutical benefits. As a result of a human error and deficiencies in its computer systems design, a company had to pay substantial damages for delivering 93,000 barrels of oil to the wrong consignee. Increasingly, it appears that organizations will be held liable for damages that occur as a result of errors in the design, implementation, or operation of their computer systems.

1.2.6 Maintenance of Privacy

Much data is now collected about us as individuals: taxation, credit, medical, educational, employment, residence, and so on. This data was also collected before computers. Nonetheless, the powerful data processing capabilities of computers, particularly their rapid throughput, integration, and retrieval capabilities, cause many people to wonder whether the privacy of individuals (and organizations) has now been eroded beyond acceptable levels. In the United States, for example, civil rights activists have long held

substantial concerns about using computer systems for computer-matching purposes (Shattuck 1984). In computer matching, disparate files are merged or compared to build up a profile on a person. A person's taxation data might be compared with data on the social security benefits they receive to detect possible instances of welfare fraud.

Similarly, in Australia, many people were concerned when the then-federal government proposed that it would introduce an Australia card (Graham 1990). Privacy activists argued that use of the Australia card as a universal identifier for each Australian would significantly undermine individual privacy.

More recently, some people have been concerned about the establishment of human genome data banks and the potential to use computers in conjunction with human genetic data to obtain detailed information about a person. They are concerned that knowledge about a person's genetics could be used in decisions about them for example, whether to give them a job or whether to give them life insurance. Also, some people are concerned about the impact of the Internet on personal privacy (Meeks 1997). For example, they fear that search engines could be used to extract data from large databases that could compromise a person's privacy.

Aside from any constitutional aspect, many nations deem privacy to be human right. These nations consider it to be the responsibility of those people concerned with computer data processing to ensure that computer use does not evolve to the stage where data about people can be collected, integrated, and retrieved quickly. Furthermore, they consider that computer professionals of all kinds have a responsibility to ensure that data is used only for the purposes intended. Unfortunately, there are now many instances in which computers have been used to abuse the privacy of individuals. As a result, computer professionals are now coming under increasing pressure to ensure that this does not happen.

1.2.7 Controlled Evolution of Computer Use

From time to time, major conflicts arise over how computer technology should be used in our societies. For example, some computer scientists continue to b concerned about using computers to support nuclear weapons command and control systems. Many became especially vocal during the debate over the U.S Strategic Defense Initiative's battle management systems. They argue that the reliability of complex computer systems usually cannot be guaranteed and that the consequences of using unreliable computer systems can be catastrophic.

Similarly, many people are concerned about the effects that use of computers can have on a person's working life. Should computer technology be allowed to displace people from the workforce or to stultify jobs? What effects do computers have on the physical and mental well-being of their users?

It might be argued that technology is neutral it is neither good nor bad. The use of technology, however, can produce major social problems. In this light, important, ongoing decisions must be made about how computers should be used in our societies. Governments, professional bodies, pressure groups, organizations, and individual persons all must be concerned with evaluating and monitoring how we deploy computer technology.

1.3 Information systems auditing defined

Information systems auditing is the process of collecting and evaluating evidence to determine whether a computer system safeguards assets, maintains data integrity, allows organizational goals to be achieved effectively, and uses resources efficiently. Thus, information systems auditing supports traditional audit objectives: attest objectives (those of the external auditor) that focus on asset safeguarding and data integrity and management objectives (those of the internal auditor) that encompass not only attest objectives but also effectiveness and efficiency objectives.

Sometimes information systems auditing has another objective namely ensuring that an organization complies with some regulation, rule, or condition. For example, a bank might have to comply with a government regulation about how much it can lend; an introduction agency might seek to comply with a voluntary code in relation to use of personal data about its clients; or an organization might seek to comply with a covenant in a loan contract that it has with a merchant bank.

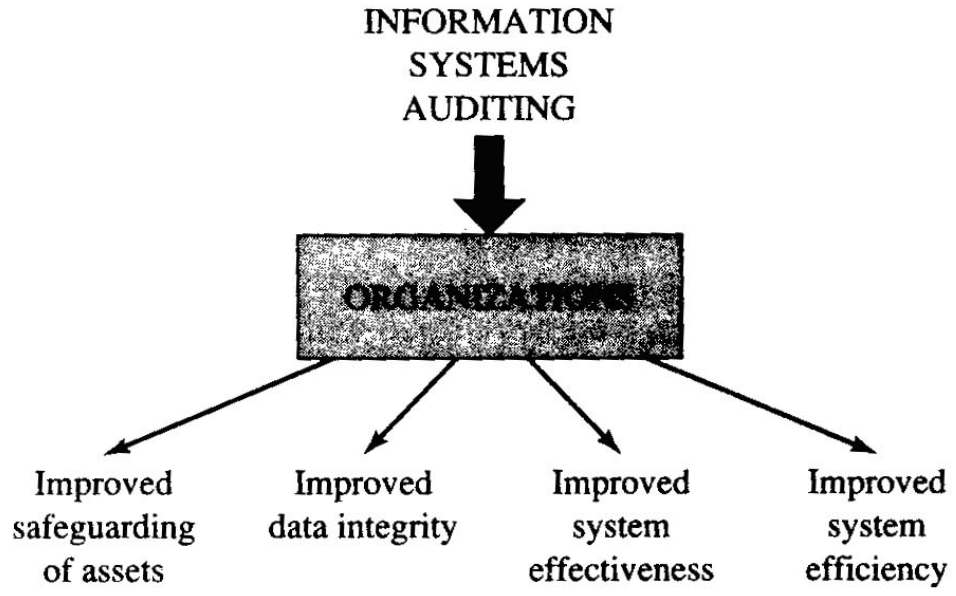


FIGURE 1-2 Impact of the information systems audit function on organizations.

However, we will not pursue the compliance auditing objective further, for the following reasons: First, we can conceive of compliance concerns within the broader framework of effectiveness objectives. (One goal of an organization is to comply with regulations, rules, and conditions to which it is subject either voluntarily or involuntarily.) Second, in any event, our broad treatment of control and audit within a computer environment should prove useful to auditors who must carry out compliance work. Third, at least in principle the notion of compliance work is straightforward, and it can be incorporated fairly easily.

In this light, Figure 1-2 shows that throughout this book we conceive information systems auditing as being a force that enables organizations to better achieve four major objectives. In the following subsections, we consider each of these objectives in detail.

1.3.1 Asset Safeguarding Objectives

The information system assets of an organization include hardware, software, facilities, people (knowledge), data files, system documentation, and supplies. Like all assets, they must be protected by a system of internal control. Hardware can be damaged maliciously. Proprietary software and the contents of data files can be stolen or destroyed. Supplies of negotiable forms can be used for unauthorized purposes. These assets are often concentrated in one or a small number of locations, such as a single disk. As a result, asset safeguarding becomes an especially important objective for many organizations to achieve.

1.3.2 Data Integrity Objectives

Data integrity is a fundamental concept in information systems auditing. It is a state implying data has certain attributes: completeness, soundness, purity, and veracity. If data integrity is not maintained, an organization no longer has a true representation of itself or of events. Moreover, if the integrity of an organization's data is low, it could suffer from a loss of competitive advantage. Nonetheless, maintaining data integrity can be achieved only at a cost. The benefits obtained should exceed the costs of the control procedures needed.

Three major factors affect the value of a data item to an organization and thus the importance of maintaining the integrity of that data item:

1. The value of the informational content of the data item for individual decision makers: The informational content of a data item depends on its ability to change the level of uncertainty surrounding a decision and, as a result, to change the expected payoffs of the decisions that might be made. These notices have been well developed within statistical decision theory.

2. The extent to which the data item is shared among decision makers: If data is shared, corruption of data integrity affects not just one user but many. The value of a data item is some aggregate function of the value of

the data item to the individual users of the data item. Thus, maintenance of data integrity becomes more critical in a shared data environment.

3. The value of the data item to competitors: If a data item is valuable to a competitor, its loss might undermine an organization's position in the marketplace. Competitors could exploit the informational content of the data item to reduce the profitability of the organization and to bring about bankruptcy, liquidation, takeover, or merger.

1.3.3 System Effectiveness Objectives

An effective information system accomplishes its objectives. Evaluating effectiveness implies knowledge of user needs. To evaluate whether a system reports information in a way that facilitates decision making by its users, auditors must know the characteristics of users and the decision-making environment.

Effectiveness auditing often occurs after a system has been running for some time. Management requests a postaudit to determine whether the system is achieving its stated objectives. This evaluation provides input to the decision on whether to scrap the system, continue running it, or modify it in some way.

Effectiveness auditing also can be carried out during the design stages of a system. Users often have difficulty identifying or agreeing on their needs. Moreover, substantial communication problems often occur between system designers and users. If a system is complex and costly to implement, management might want auditors to perform an independent evaluation of whether the design is likely to fulfill user needs.

1.3.4 System Efficiency Objectives

An efficient information system uses minimum resources to achieve its required objectives. Information systems consume various resources: machine time, peripherals, system software, and labor. These resources are scarce, and different application systems usually compete for their use.

The question of whether an information system is efficient often has no clear-cut answer. The efficiency of any particular system cannot be considered in isolation from other systems. Problems of suboptimization occur if one system is "optimized" at the expense of other systems. For example, minimizing an application system's execution time might require dedication of some hardware resource (e.g., a printer) to that system. The system might not use the hardware fully, however, while it undertakes its work. The slack resource will not be available to other application systems if it is dedicated to one system.

System efficiency becomes especially important when a computer no longer has excess capacity. The performance of individual application systems degrades (e.g., slower response times occur), and users can become increasingly frustrated. Management must then decide whether efficiency

can be improved or extra resources must be purchased. Because extra hardware and software is a cost issue, management needs to know whether available capacity has been exhausted because individual application systems are inefficient or because existing allocations of computer resources are causing bottlenecks. Because auditors are perceived to be independent, management might ask them to assist with or even perform this evaluation.

1.4 Effects of computers on internal controls

The goals of asset safeguarding, data integrity, system effectiveness, and system efficiency can be achieved only if an organization's management sets up a system of internal control. Traditionally, major components of an internal control system include separation of duties, clear delegation of authority and responsibility, recruitment and training of high-quality personnel, a system of authorizations, adequate documents and records, physical control over assets and records, management supervision, independent checks on performance, and periodic comparison of recorded accountability with assets. In a computer system, these components must still exist; however, use of computers affects the implementation of these internal control components in several ways. In essence, they have been adopted and adapted to fit in with a computer environment (Figure 1-3). In the following subsections, we briefly examine some of the major areas of impact.

1.4.1 Separation of Duties

In a manual system, separate persons should be responsible for initiating transactions, recording transactions, and maintaining custody of assets. As a basic control, separation of duties prevents or detects errors and irregularities. In a computer system, however, the traditional notion of separation of duties does not always apply. For example, a program could reconcile a vendor invoice against a receiving document and print a check for the amount owed to a creditor. Thus, the program is performing functions that in manual systems would be considered to be incompatible. Nevertheless, it might be inefficient and, from a control viewpoint, useless to place these functions in separate programs. Instead, separation of duties must exist in a different form. When it has been determined that the program executes correctly, the capability to run the program in production mode and the capability to change the program must be separated.

In minicomputer and microcomputer environments, separation of incompatible functions could be even more difficult to achieve. Some minicomputers and microcomputers allow users to change programs and data easily. Furthermore, they might not provide a record of these changes. Thus, determining whether incompatible functions have been performed by system users can be difficult or impossible.

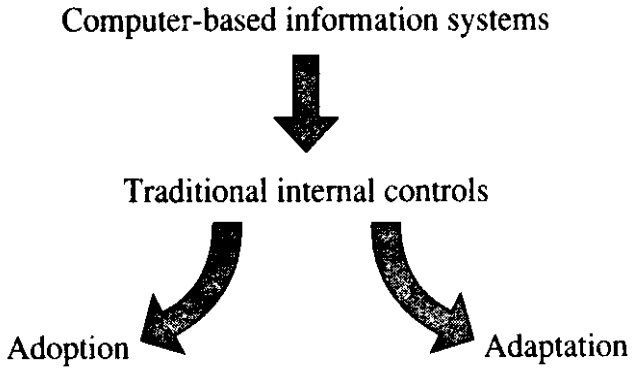


FIGURE 1-3 Effects of computer –based information systems on traditional internal controls.

1.4.2 Delegation of Authority and Responsibility

A clear line of authority and responsibility is an essential control in both manual and computer systems. In a computer system, however, delegating authority and responsibility in an unambiguous way might be difficult because some resources are shared among multiple users. For example, one objective of using a database management system is to provide multiple users with access to the same data, thereby reducing the control problems that arise with maintaining redundant data. When multiple users have access to the same data and the integrity of the data is somehow violated, it is not always easy to trace who is responsible for corrupting the data and who is responsible for identifying and correcting the error. Some organizations have attempted to overcome these problems by designating a single user as the owner of the data. This user assumes ultimate responsibility for the integrity of the data.

Authority and responsibility lines have also been blurred by the rapid growth in end-user computing. Because high-level languages are more readily available, more users are developing, modifying, operating, and maintaining their own application systems instead of having this work performed by information systems professionals. Although these developments have substantial benefits for the users of computing services in an organization, unfortunately they exacerbate the problems of exercising overall control over computing use.

1.4.3 Competent and Trustworthy Personnel

Substantial power is often vested in the persons responsible for the computer based information systems developed, implemented, operated, and maintained within organizations. For example, a systems analyst might be responsible for advising management on the suitability of high-cost, high-technology equipment. Similarly, a computer operator sometimes takes

responsibility for safeguarding critical software and critical data during execution of or backup of a system. The power vested in the personnel responsible for computer systems often exceeds the power vested in the personnel responsible for manual systems.

Unfortunately, ensuring that an organization has competent and trustworthy information systems personnel is a difficult task. In many countries and across many years, well-trained and experienced information systems personnel have been in short supply. Therefore, organizations sometimes have been forced to compromise in their choice of staff. Moreover, it is not always easy for organizations to assess the competence and integrity of their information systems staff. High turnover among these staff has been the norm. Therefore, managers have had insufficient time to evaluate them properly. In addition, the rapid evolution of technology inhibits management's ability to evaluate an information systems employee's skills. Some information systems personnel also seem to lack a well-developed sense of ethics, and some seem to delight in subverting controls.

1.4.4 System of Authorizations

Management issues two types of authorizations to execute transactions. First, general authorizations establish policies for the organization to follow. For example, a fixed price list is issued for personnel to use when products are sold. Second, specific authorizations apply to individual transactions. For example acquisitions of major capital assets might have to be approved by the board of directors.

In a manual system, auditors evaluate the adequacy of procedures for authorization by examining the work of employees. In a computer system, authorization procedures often are embedded within a computer program. For example, the order-entry module in a sales system might determine the price to be charged to a customer. Thus, when evaluating the adequacy of authorization procedures, auditors must examine not only the work of employees but also the veracity of program processing.

In a computer system it is also more difficult to assess whether the authority assigned to individual persons is consistent with management's wishes. For example, it might be hard to determine exactly what data users can access when they are provided with a generalized retrieval language. Users might be able to formulate queries on a database in such a way that they could infer the contents of confidential information. Indeed, substantial research is now being undertaken on controls that prevent violation of the privacy of data. Chapter 15 examines inference controls in more detail.

1.4.5 Adequate Documents and Records

In a manual system, adequate documents and records are needed to provide an audit trail of activities within the system. In computer systems, documents might not be used to support the initiation, execution, and recording of some transactions. For example, in an online order-entry system, customers' orders received by telephone might be entered directly

into the system. Similarly, some transactions might be activated automatically by a computer system. For example, an inventory replenishment program could initiate purchase orders when stock levels fall below a set amount. Thus, no visible audit or management trail would be available to trace the transaction.

The absence of a visible audit trail is not a problem for auditors, provided that systems have been designed to maintain a record of all events and the record can be easily accessed. In well-designed computer systems, audit trails are often more extensive than those maintained in manual systems. Unfortunately, not all computer systems are well designed. Some software, for example, does not provide adequate access controls and logging facilities to ensure preservation of an accurate and complete audit trail. When this situation is coupled with a decreased ability to separate incompatible functions, serious control problems can arise.

1.4.6 Physical Control over Assets and Records

Physical control over access to assets and records is critical in both manual systems and computer systems. Computer systems differ from manual systems, however, in the way they concentrate the information systems assets and records of an organization. For example, in a manual system, a person wishing to perpetrate a fraud might need access to records that are maintained at different physical locations. In a computer system, however, all the necessary records can be maintained at a single site namely, the site where the computer is located. Thus, the perpetrator does not have to go to physically disparate locations to execute the fraud.

This concentration of information systems assets and records also increases the losses that can arise from computer abuse or a disaster. For example, a fire that destroys a computer room could result in the loss of all major master files in an organization. If the organization does not have suitable backup, it might be unable to continue operations.

1.4.7 Adequate Management Supervision

In a manual system, management supervision of employee activities is relatively straightforward because the managers and the employees are often at the same physical location. In computer systems, however, data communications facilities can be used to enable employees to be closer to the customers they service. Thus, supervision of employees might have to be carried out remotely. Supervisory controls must be built into the computer system to compensate for the controls that usually can be exercised through observation and inquiry.

Computer systems also make the activities of employees less visible to management. Because many activities are performed electronically, managers must periodically access the audit trail of employee activities and examine it for unauthorized actions. Again, the effectiveness of observation and inquiry as controls is decreased.

1.4.8 Independent Checks on Performance

In manual systems, independent checks are carried out because employees are likely to forget procedures, make genuine mistakes, become careless, or intentionally fail to follow prescribed procedures. Checks by an independent person help to detect any errors or irregularities. If the program code in a computer system is authorized, accurate, and complete, the system will always follow the designated procedures in the absence of some other type of failure tike a hardware or systems software failure. Thus, independent checks on the performance of programs often have little value. Instead, the control emphasis shifts to ensuring the veracity of program code. Insofar as many independent checks on performance are no longer appropriate, auditors must now evaluate the controls established for program development, modification, operation, and maintenance.

1.4.9 Comparing Recorded Accountability with Assets

Data and the assets that the data purports to represent should periodically be compared to determine whether incompleteness or inaccuracies in the data exist or whether shortages or excesses in the assets have occurred. In a manual system, independent staffs prepare the basic data used for comparison purposes. In a computer system, however, software is used to prepare this data. For example, a program can be implemented to sort an inventory file by warehouse location and to prepare counts by inventory item at the different warehouses. li unauthorized modifications occur to the program or the data files that the program uses, an irregularity might not be discovered for example, pilfering of inventory from a particular warehouse bin. Again, internal controls must be implemented to ensure the veracity of program code, because traditional separation of duties no longer applies to the data being prepared for comparison purposes.

1.5 Effects of computers on auditing

When computer systems first appeared, many auditors were concerned that the fundamental nature of auditing might have to change to cope with the new technology. It is now clear this is not the case. Auditors must still provide a competent, independent evaluation as to whether a set of economic activities has been recorded and reported according to established standards or criteria,

Nevertheless, computer systems have affected how auditors carry out their two basic functions: evidence collection and evidence evaluation. We examine some of these changes in the following subsections.

1.5.1 Changes to Evidence Collection

Collecting evidence on the reliability of a computer system is often more complex than collecting evidence on the reliability of a manual system. Auditors confront a diverse and sometimes complex range of internal control technology that did not exist in manual systems. For example, accurate and complete operation of a disk drive requires a set of hardware controls not

used in a manual system. Similarly, system development controls include procedures for testing programs that would not be found in the development of manual systems. Auditors must understand these controls if they are to be able to collect evidence competently on the reliability of the controls.

Unfortunately, understanding the control technology is not always easy. Hardware and software continue to evolve rapidly. Although some time lag occurs, the associated controls evolve rapidly also. For example, with increasing use of data communications for data transfer, substantial research continues to be undertaken on the development of cryptographic controls to protect the privacy of data. Auditors must keep up with these developments if they are to be able to evaluate the reliability of communications networks competently.

The continuing evolution of control technology also makes it more difficult for auditors to collect evidence on the reliability of controls. Indeed, in some cases auditors might be unable to collect audit evidence using manual means. Thus, they need computer systems themselves if they are to be able to collect the necessary evidence. The development of generalized audit software occurred, for example, because auditors needed access to data maintained on magnetic media. Similarly, new audit tools might be required in due course if auditors are to be able to evaluate the reliability of controls in data communications networks competently. Unfortunately, the development of these audit tools usually lags the development of the technology that must be evaluated. In the meantime, auditors are often forced to compromise in some way when performing the evidence collection function.

1.5.2 Changes to Evidence Evaluation

Given the increased complexity of computer systems and internal control technology, it is also more difficult to evaluate the consequences of control strengths and weaknesses for the overall reliability of systems. First, auditors must understand when a control is acting reliably or malfunctioning. Next, they must be able to trace the consequences of the control strength or weakness through the system. In a shared data environment, for example, this task might be difficult. A single input transaction could update multiple data items that are used by diverse, physically disparate users. Somehow auditors must be able to trace the consequences of an error in the transaction input for all users.

In some ways, auditors are also under greater stress when they perform the evidence evaluation function for computer systems. As noted earlier, the consequences of errors in a computer system can be more serious than the consequences of errors in a manual system. Errors in manual systems tend to occur stochastically; for example, periodically a clerk prices an inventory item incorrectly. Errors in computer systems tend to be deterministic; for example, an erroneous program always will execute incorrectly. Moreover, errors are generated at high speed, and the cost to correct and rerun programs can be high. Whereas fast feedback can be provided to clerks if

they make errors, errors in computer programs can involve extensive redesign and reprogramming. Thus, internal controls that ensure that high-quality computer systems are designed, implemented, operated, and maintained are critical. The onus is on auditors to ensure that these controls are sufficient to maintain asset safeguarding, data integrity, system effectiveness, and system efficiency and that they are in place and working reliably.

1.6 Foundations of information systems auditing

Information systems auditing is not just a simple extension of traditional auditing. Recognition of the need for an information systems audit function comes from two directions. First, auditors realized that computers had affected their ability to perform the attest function. Second, both corporate and information systems management recognized that computers were valuable resources that needed controlling like any other key resource within an organization.

Figure 1-4 shows that the discipline of information systems auditing has been shaped by knowledge obtained from four other disciplines: traditional auditing, information systems management, behavioral science, and computer science.

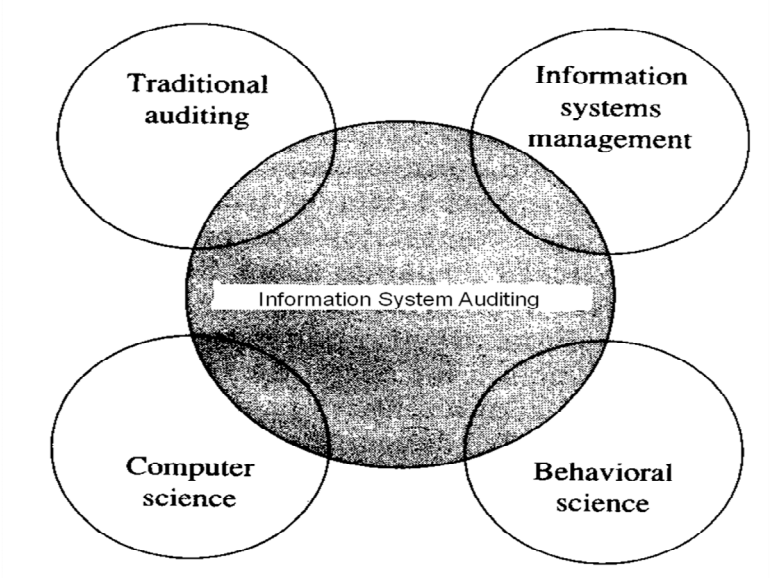


FIGURE 1-4 Information systems auditing as an intersection of other disciplines

1.6.1 Traditional Auditing

Traditional auditing brought to information systems auditing a wealth of knowledge and experience with internal control techniques. This knowledge and experience has had an impact on the design of both the manual and machine components of an information system.

For example, in a computer system, clerical activities, such as data preparation activities, are often a critical component of the system. As with manual systems, these activities should be subject to internal control principles such as separating incompatible duties, having competent and trustworthy personnel, and establishing clear definitions of duties. By applying these principles, management seeks to ensure that the integrity of data is maintained before it is entered into the computer-based components of the information system.

Similarly, traditional auditing concepts like control totals are also relevant to the update and maintenance of files by computer programs. Computer pro grams must ensure that all transaction data are processed and that they are processed correctly. Control totals have had longstanding use in information systems because these concerns also exist when humans (rather than programs) update and maintain files.

The general methodologies for evidence collection and evidence evaluation used by information system auditors are also based on traditional auditing methodologies. The long evolution of and extensive experience gained in traditional auditing highlight the critical importance of having objective, verifiable evidence and an independent evaluation of information systems.

Perhaps most important, traditional auditing brings to information systems auditing a control philosophy. It is difficult to articulate the nature of this philosophy. One can glean elements, however, by reading auditing literature or examining the work of auditors. The philosophy involves examining information systems with a critical mind, always with a view to questioning the capability of an information system to safeguard assets, maintain data integrity, and achieve its objectives effectively and efficiently.

1.6.2 Information Systems Management

The early history of computer-based information systems shows some spectacular disasters. Massive cost overruns occurred, and many systems failed to achieve their stated objectives. As a result, for many years researchers have been concerned with identifying better ways of managing the development and implementation of information systems.

Some important advances have been made. For example, techniques of project management have been carried across into the information systems area with considerable success. Documentation, standards, budgets, and variance investigation are now emphasized. Better ways of developing and implementing systems have been developed. For example, object-oriented analysis, design, and programming have enabled programmers to develop software faster, with fewer errors and easier maintenance characteristics. These advances affect information systems auditing because they ultimately affect asset safeguarding, data integrity, system effectiveness, and system efficiency objectives.

2. Conducting an information systems audit

Structure

Objective

In this lesson you will learn about:

- information system audit controls
- dealing with audit complexity and risk factors
- types of audit procedures
- planning of audit control

2.1 Introduction

It is a sobering experience to be in charge of the information systems audit of an organization that has several hundred programmers and analysts, many computers, and thousands of files. Obviously, all organizations are not this size. Except for the smallest organizations, however, auditors usually

cannot perform a detailed check of all the data processing carried out within the information systems function. Instead, they must rely on a sample of data to determine whether the objectives of information systems auditing are being achieved.

How, then, can we perform information systems audits so that we obtain reasonable assurance that an organization safeguards its data-processing assets, maintains data integrity, and achieves system effectiveness and efficiency? To address this question, this chapter provides an overview of a general approach that we can use to conduct an information systems audit.

We start by examining the nature of controls and discussing some techniques for simplifying and providing order to the complexity encountered when .making evaluation judgments on computer-based information systems. Next we consider some of the basic risks auditors' face, how these risks affect the overall approach to an audit, and the types of audit procedures used to assess or control the level of these risks. We then consider the basic steps to be undertaken in the conduct of an information systems audit. Finally, we examine a major decision auditors must make when planning and conducting an information systems audit namely, how much do they need to know about the internal workings of a computer-based information system before an effective audit can be conducted?

2.2 The nature of controls

Information systems auditors ultimately are concerned with evaluating the reliability, or operating effectiveness, of controls. It is important, therefore, that we understand what is meant by a control.

A control is a system that prevents, detects, or corrects unlawful events. There are three key aspects to this definition.

First, a control is a system. In other words, it comprises a set of interrelated components that function together to achieve some overall purpose. Unfortunately, we tend to name controls by focusing on just one feature of the control. For example, probably all of us are familiar with a password control. A password, per se, however, is not a control. Passwords become a control only in the context of a system that allows secure issue of or choice of passwords, correct validation of passwords, secure storage of passwords, follow-up on illicit use of passwords, and so on. If this system breaks down in some way, passwords will be ineffective as a control. In short, the term "password control" is a notation for the constellation of things that work together to ensure only authorized people use computing resources. When we evaluate a control, therefore, we must consider its reliability from a systems perspective.

Second, the focus of controls is unlawful events. An unlawful event can arise if unauthorized, inaccurate, incomplete, redundant, ineffective, or inefficient input enters the system. For example, a data-entry clerk might key incomplete data into the system. An unlawful event can also arise if the system

transforms the input in an unauthorized, inaccurate, incomplete, redundant, ineffective, or inefficient way. For example, a program could contain erroneous instructions that result in incorrect computations being performed. Whatever the reason, the system moves into a state that we deem to be unacceptable.

Third, controls are used to prevent, detect, or correct unlawful events. Consider some examples:

1. Preventive control: Instructions are placed on a source document to prevent clerks from filling it out incorrectly. Note that the control works only if the instructions are sufficiently clear and the clerk is sufficiently well trained to understand the instructions. Thus, both the clerk and the instructions are components of the system that constitutes the control. The instructions by themselves are not the control.

2. Detective control: An input program identifies incorrect data entered into a system via a terminal. Again, the control is a system because various parts of the program must work together to pinpoint errors.

3. Corrective control: A program uses special codes that enable it to correct data corrupted because of noise on a communications line. Once more, the control is a system because various parts of the program must work together in conjunction with the error-correcting codes to rectify the error.

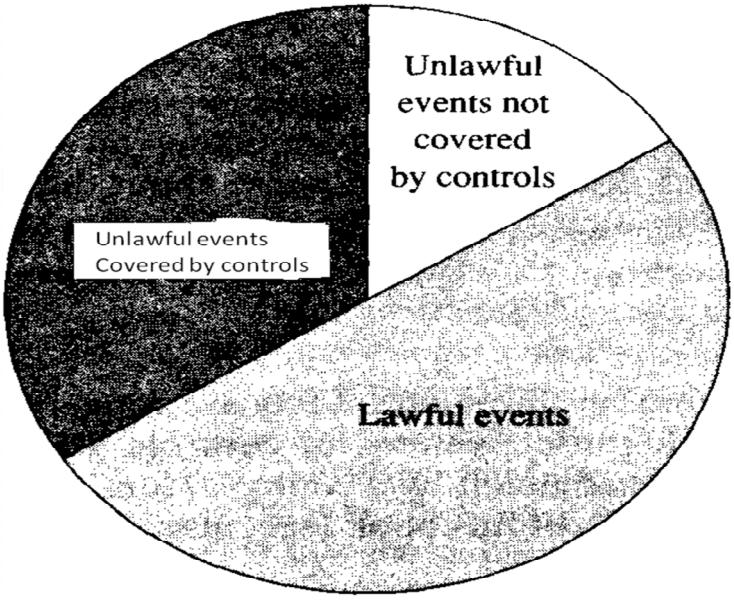


FIGURE 2-1 Lawful and unlawful events in an information system

The overall purpose of controls is to reduce expected losses from unlawful events that can occur in a system. They do so in two ways. First, preventive controls reduce the probability of unlawful events occurring in the first place.

For example, instructions on a source document reduce the likelihood of the clerk who completes the document making an error. Second, detective and corrective controls reduce the amount of the losses that arise if the unlawful event occurs. For example, if a data-entry clerk keys incorrect data into a computer system, an input validation control might detect that the data is in error and halt further processing. A small loss arises from delayed processing, but larger losses associated with a corrupted database do not occur. In addition, the control might be able to determine the nature of the keying error made, perhaps on the basis of past keying errors, and correct the error without the clerk having to intervene. Thus, the losses associated with recovering from the error are also reduced.

The auditor's task is to determine whether controls are in place and working to prevent the unlawful events that might occur within a system. Auditors must be concerned to see that at least one control exists to cover each unlawful event that might occur. Usually, some unlawful events in a system will not be covered because a cost-effective control cannot be found (Figure 2-1). Even if an unlawful event is covered by a control, however, auditors must evaluate whether the control is operating effectively. Moreover, if more than one control covers an unlawful event (i.e., redundant controls exist), auditors must ensure that all operate effectively. Otherwise, losses can be incurred because of reliance on a malfunctioning control instead of a reliable one.

2.3 Dealing with complexity

Conducting an information systems audit is an exercise in dealing with complexity. Auditors somehow must accomplish their objectives given the myriad of systems. Because complexity is a root cause of the problems faced by many professionals (e.g., engineers, architects), researchers have attempted to develop guidelines that reduce complexity (see, e.g., Simon-1981). In the following subsections we consider two major guidelines that underlie the approach taken when conducting an information systems audit:

1. Given the purposes of the information systems audit, factor the system to be evaluated into subsystems.

2. Determine the reliability of each subsystem and the implications of each subsystem's level of reliability for the overall level of reliability in the system.

2.3.1 Subsystem Factoring

The first step in understanding a complex system is breaking it up into subsystems. A subsystem is a component of a system that performs some basic function needed by the overall system to enable it to attain its fundamental objectives. Subsystems are logical components rather than physical components. In other words, you cannot "touch" a subsystem. It exists only in the eye of the beholder. For example, we cannot see the input subsystem in a computer system. Instead, we see such things as terminals

and data-entry clerks that function to get data into the system, but these things are components of the input subsystem and not the subsystem itself.

The process of decomposing a system into subsystems is called factoring. Factoring is an iterative process that terminates when we feel we have broken down the system into parts small enough to be understood and evaluated. In other words, each subsystem is decomposed into its constituent subsystems, which, in turn, are decomposed again until we can sufficiently comprehend the subsystem with which we are dealing. The system to be evaluated can then be described as a level structure of subsystems, with each subsystem performing a function needed by some higher-level subsystem (Figure 2-2).

To undertake the factoring process, we need some basis for identifying subsystems. One basis has been suggested already: The essence of a subsystem is the function it performs. Auditors should look first; therefore, for the fundamental functions a system performs to accomplish its overall objectives. Different functions delineate different subsystems. For example, the overall objective of some types of organizational systems is to make a profit. One critical function that must be performed in these systems is the receipt of customer orders. This function delineates the order-entry subsystem, which is distinct from, say, the subsystem that receives and processes customer payments as its basic function. The order-entry subsystem, in turn, can be broken down into further subsystems. These lower-level subsystems are defined on the basis of (sub)functions that must be performed to accomplish the overall objective of getting orders recorded accurately and completely. For example, functions are required to check whether sufficient inventory is available to satisfy an order and determine whether a customer's credit limit has been exceeded.

Besides function, systems theory indicates that two other guidelines should underlie the way in which we identify and delineate subsystems.

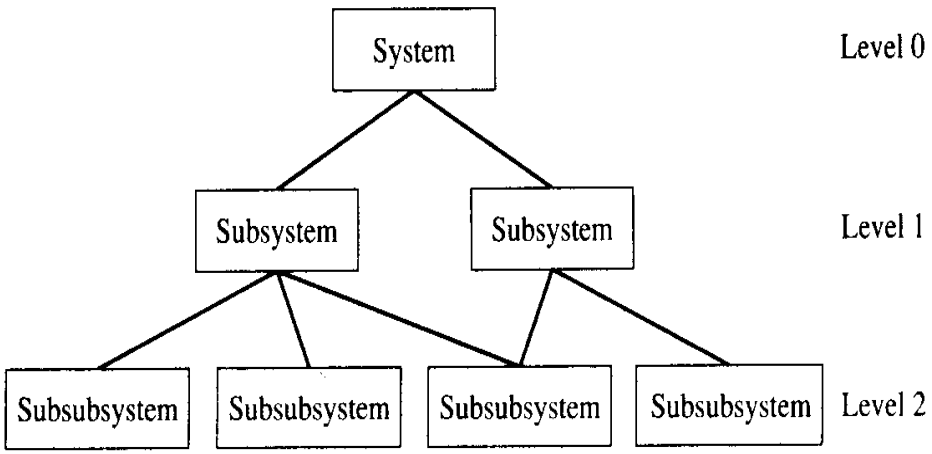


FIGURE 2-2 Level structure of systems and subsystems

First each subsystem should be relatively independent of other subsystems. The objective is for each subsystem to be loosely coupled to other subsystems. If this objective can be achieved, auditors can evaluate the subsystem in relative isolation from other subsystems. In other words, to some extent auditors can disregard the effects of control strengths and weaknesses in other systems.

Second, each subsystem should be internally cohesive. All the activities performed by the subsystem should be directed toward accomplishing a single function. If this objective can be achieved, it will be easier for auditors to understand and evaluate the activities carried out by the subsystem.

The theory of coupling and cohesion has been extensively developed. From an audit viewpoint, however, the pragmatic issue is that subsystems are difficult to understand and their reliability is difficult to evaluate unless they are loosely coupled with other subsystems and internally cohesive. An understanding of complex systems can only be obtained if each of their parts can be studied relatively independently and the activities performed by each part are clear. When we decompose a system into subsystems, therefore, we should evaluate the extent of coupling and cohesion in the subsystems we choose. If the subsystems are not loosely coupled and internally cohesive, we should attempt a different factoring. If no factoring seems to delineate subsystems that possess these characteristics, we will have difficulty evaluating the reliability of the system because its activities are too convoluted. Indeed, auditors have long recognized that some systems cannot be audited. The theory of coupling and cohesion provides the underlying rationale for this conclusion when such systems are encountered.

At least conceptually, auditors might choose to factor systems in several different ways. Over time, however, auditors have found two ways to be especially useful when conducting information systems audits (Figure 2-3). The first is according to the managerial functions that must be performed to ensure that development, implementation, operation, and maintenance of information systems proceed in a planned and controlled manner. Managerial systems function to provide a stable infrastructure in which information systems can be built, operated, and maintained on a day-to-day basis. Several types of management subsystems have been identified that correspond to the organizational hierarchy and some of the major tasks performed by the information systems function:

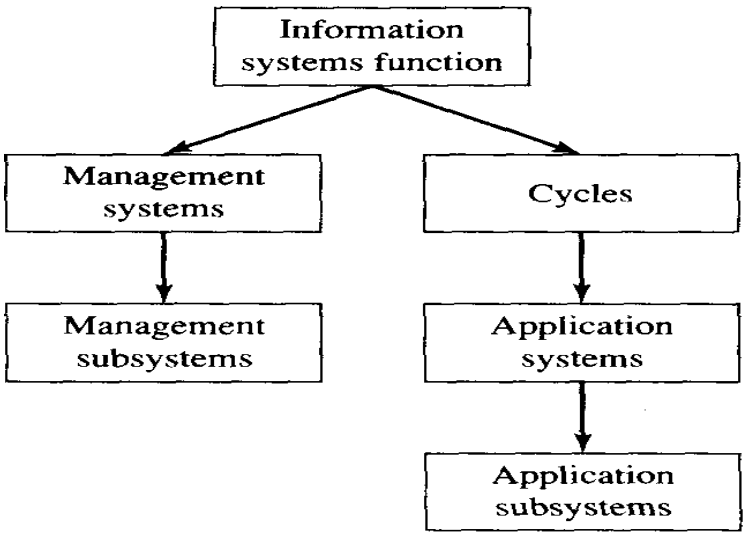


FIGURE 2-3. Decomposition of the information-systems function

Management Subsystem	*Description of Subsystem*
Top management	Top management must ensure the information systems function is well managed. It is responsible primarily for long-run policy decisions on how information systems will be used in the organization.
Information systems management	Information systems management has overall responsibility for the planning and control of all information systems activities. It also provides advice to top management in relation to long-run policy decision making and translates long-run policies into short-run goals and objectives.
Systems development management	Systems development management is responsible for the design, implementation, and maintenance of application systems.
Programming management	Programming management is responsible for programming new systems, maintaining old systems, and providing general systems support software.
Data administration	Data administration is responsible for addressing planning and control issues in relation to use of an organization's data.
Quality assurance management	Quality assurance management is responsible for ensuring information systems development, implementation, operation, and maintenance conform to established quality standards.
Security administration	Security administration is responsible for access controls and physical security over the information systems function.
Operations management	Operations management is responsible for planning and control of the day-to-day operations of information systems.

The second factoring is according to the application functions that need to be undertaken to accomplish reliable information processing. This factoring corresponds to the "cycles" approach auditors have traditionally used to conduct an audit. The information systems supporting an organization are first grouped into cycles. These cycles vary across industries, but a typical set for a commercial or manufacturing enterprise includes (a) sales and collections, (b) payroll and personnel, (c) acquisitions and payments, (d) conversion, inventory, and warehousing, and (e) treasury. Each cycle is then factored into one or more application systems. For example, the sales and collections cycle comprises an order-entry application system, a billing application system, and an accounts-receivable application system. Application systems, in turn, are then factored into subsystems. The set of application subsystems includes the following:

APPLICATION SYSTEM	DESCRIPTION OF SUBSYSTEM
Boundary	Comprises the components that establish the interface between the user and the system
Input	Comprises the components that capture, prepare, and enter commands and data into the system
Communications	Comprises components that transmit data among subsystems and systems
Processing	Comprises the components that perform decision making, computation, classification, ordering, and summarization of data in the system
Database	Comprises the components that define, add, access, modify, and delete data in the system
Output	Comprises the components that retrieve and present data to users of the system

Neither of these two types of decomposition is irrevocable, and in due course others might prove better. Nevertheless, they currently underlie the audit approaches advocated by many professional bodies of auditors. They allow us to decrease complexity to a point where we can understand and evaluate the nature of and reliability of subsystems. Throughout the rest of this book, these two factorings will be used in our examination of information systems controls.

2.3.2 Assessing Subsystem Reliability

After we have identified the lowest-level subsystems in our level structure of subsystems, we can evaluate the reliability of controls. Beginning with the lowest-level subsystems, we first attempt to identify all the different types of events that might occur in these subsystems. We must be mindful of both the lawful events and the unlawful events that can occur. Nevertheless, an auditor's primary concern will be with any unlawful events that might arise.

As a basis for identifying lawful and unlawful events in management subsystems, we focus on the major functions each subsystem performs. We consider how each function should be undertaken and then evaluate how well a subsystem complies with our normative views. For example, an important function that should be performed by the top-management subsystem is information systems planning. Given the nature of the organization we are auditing, we might determine that top management should undertake extensive strategic planning but only a moderate level of operational planning if the long-run future of information systems within the organization is to be ensured. These views form the basis for determining which information systems planning events are to be deemed lawful and which are to be deemed unlawful. We then identify the information systems planning events that have occurred and classify them as either lawful or unlawful. For example, if no strategic planning has been undertaken, an unlawful event has occurred. Failure to plan ultimately undermines asset safeguarding, data integrity, system effectiveness, and system efficiency objectives. Similarly, if too much operational planning has been undertaken, an unlawful event has occurred because resources have been wasted. As a result, system effectiveness and system efficiency objectives have been undermined.

Perhaps the key aspect of identifying lawful and unlawful events in management subsystems is the decision of how a particular function should be performed within the subsystem. After substantial research on information systems management, it is now clear that the way information systems management functions should be performed in organizations must vary, depending on the particular circumstances faced by each organization. For example, in some organizations, strategic information systems planning are a critical function, but in others it has only minor importance. Auditors must be knowledgeable and astute in determining the ways that management functions should be performed in each organization evaluated. Otherwise, judgments on what events are lawful will be misguided.

As a basis for identifying lawful and unlawful events in application subsystems, we focus on the transactions that can occur as input to the subsystem. All events in an application system must arise from a transaction. The application system initially changes state (an event occurs) when the transaction is first received as an input. For example, an order-entry system must record an order when it is first entered into the system. Further state changes (events) then occur as the application system processes the transaction. For example, after an order-entry system has

stored an open order, it then attempts to fill the order. Lawful events will arise if the transaction and subsequent processing are authorized, accurate, complete, non-redundant, effective, and efficient. Otherwise, unlawful events will occur.

To identify all the events that might arise in an application system as a result of a transaction, we must understand how the system is likely to process the transaction. Historically, auditors have used walk-through techniques to accomplish this objective: They consider a particular transaction, identify the particular components in the system that process the transaction, and then try to understand each processing step that each component executes. They also consider any errors or irregularities (unlawful events) that might occur along the way. For example, auditors might focus on a credit-sale transaction. After the transaction has been entered into the sales system, they would trace the credit sale through each processing step executed by the order-entry program. They would also consider how the transaction might be entered improperly and how subsequent processing errors or irregularities might arise.

It is often costly to trace each individual transaction through an application system to obtain an understanding of all the different types of events that can occur in the system. For this reason, auditors sometimes focus on classes of transactions. In other words, they group transactions together if the transactions undergo similar processing. They then try to understand these transactions and the events that arise as a result of these transactions as a group. In addition, they focus only on those transactions they consider to be material from the viewpoint of their audit objectives. Using these strategies, not all events that can occur in a system are identified. Nevertheless, auditors should examine all those transactions and events that they consider to be important.

When the material events that can occur in a management of application system have been identified, auditors must evaluate whether controls are in place and working to cover the unlawful events. Accordingly, they collect evidence on the existence and reliability of controls to determine whether expected losses from unlawful events have been reduced to an acceptable level. They consider each type of unlawful event that might arise, whether controls cover each of these events, how reliable these controls are, and whether a material error or irregularity can still occur. Lists have been published to assist with this task, showing failings that occur in management subsystems and errors and irregularities that occur for different types of transactions in different types of application systems. These lists also show various controls that can be used to reduce expected losses from these errors and irregularities. Table is an example of one such list for a customer-order transaction in an order-entry application system. The table shows a controls matrix in which the columns show errors or irregularities that can occur and the rows show controls that can be set up to reduce expected losses from these errors and irregularities. The elements of the matrix show an auditor's assessment of how effective each control is in reducing expected losses from each type of error or irregularity.

Errors and Controls / Irregularities	Unauthorized customer	Unauthorized terms and credit	Incorrect quantity	Incorrect price	Untimely processing
Order-entry operator well trained	M	M	M	M	M
User-friendly interface			M	M	M
High-quality input validation program	H	H	H	H	
Management review of sales overrides	M	M	L	M	
Daily report on unfilled orders					H
Management review of daily transaction volumes					M

Note: H = high-effectiveness control; M = moderate-effectiveness control; L = low-effectiveness control

The evaluation of reliability proceeds upwards in the level structure of a system. Lower-level subsystems are components of higher-level systems. When the reliability of a lower-level system has been assessed, its impact on the nature of and frequency of unlawful events in higher-level systems can be evaluated. The evaluation proceeds until the highest-level system (the entire system) has been considered. For every system at every level in the level structure, the evaluation steps are the same. The transactions that might enter the system are first identified. The lawful and unlawful events that can occur as a result are then considered. Finally, the reliability of the controls that cover the unlawful events is assessed.

As we evaluate higher-level systems, we are likely to encounter new controls for three reasons. First, controls in lower-level systems can malfunction. Recall, a control is a system itself, and it can be unreliable like any other system. A higher-level control might be implemented to cover unlawful events that arise when lower-level controls fail to prevent, detect, or correct them. For example, consider a group of clerks that process mail orders. Work might be divided among them based on the first letter of customers' surnames. Thus, several subsystems exist to process orders from different groups of customers. Each clerk might exercise certain controls to prevent, detect, or correct errors. Nevertheless, their manager might also examine the quality of their work. Managers are responsible for the quality of work in all subsystems, and they are exercising a higher-level control in case a lower-level control malfunctions.

Second, it might be more cost-effective to implement controls at higher levels. Again, consider our example of the group of clerks who process mail orders. If they are well trained and diligent, they might not be required to double-check their work. Given the low error rate that is expected to occur, the cost of double-checking might be too high. Their manager periodically might take a sample of their work, however, to assess its quality. The higher-level control is more cost-effective because it is exercised by one person who has greater facility with the control rather than multiple persons, each of whom has less facility with the control.

Third, some events are not manifested as unlawful except in higher-level systems. For example, an employee might query a database to obtain the

average salary of female consultants employed within an organization. The subsystem that processes the query might deem this query to be a lawful event. The person might then query the database to obtain the number of female consultants employed within the organization. Again, the subsystem that processes the query might deem it to be lawful. If the organization employs only one female consultant, however, the employee now knows the consultant's salary. A higher-level system control is needed to detect the violation of the confidentiality of her salary. When the two lawful events are considered together, the overall event is unlawful.

Clearly, the process of aggregating subsystem reliability assessments to higher levels can be a difficult task. Errors made at one level of assessment will propagate to higher levels of assessment. Auditors must take substantial care with evidence-collection processes and evaluation judgments, especially as they begin to fix evaluation judgments in lower-level subsystems and move to higher-level subsystems and systems.

2.4 Audit risks

Recall that information systems auditors are concerned with four objectives: asset safeguarding, data integrity, system effectiveness, and system efficiency. Both external and internal auditors are concerned with whether errors or irregularities cause material losses to an organization or material misstatements in the financial information prepared by the organization. If you are an internal auditor, it is likely you will also be concerned with material losses that have occurred or might occur through ineffective or inefficient operations. External auditors, too, might be concerned when ineffective or inefficient operations threaten to undermine the organization. Moreover, many external auditors report such problems as part of their professional services to the management of an organization.

To assess whether an organization achieves the asset safeguarding, data integrity, system effectiveness, and system efficiency objectives, auditors collect evidence. Because of the test nature of auditing, auditors might fail to detect real or potential material losses or account misstatements. The risk of an auditor failing to detect actual or potential material losses or account misstatements at the conclusion of the audit is called the audit risk. Auditors choose an audit approach and design audit procedures in an attempt to reduce this risk to a level deemed acceptable.

As a basis for determining the level of desired audit risk, some professional bodies of auditors have adopted the following audit risk model for the external audit function:

$$DAR = IR \times CR \times DR$$

In this model, DAR is the desired audit risk (as discussed previously). IR is the inherent risk, which reflects the likelihood that a material loss or account misstatement exists in some segment of the audit before the reliability of internal controls is considered. CR is the control risk, which reflects the

likelihood that internal controls in some segment of the audit will not prevent, detect, or correct material losses or account misstatements that arise. DR is the detection risk, which reflects that the audit procedures used in some segment of the audit will fail to detect material losses or account misstatements.

Note that in all cases the risks incorporated into this model are defined to be those associated with the attest objectives of external auditors. We can easily broaden them, however, to include the risks associated with real or potential material losses from ineffective or inefficient operations. In other words, the model is sufficiently general to cover our four objectives for information systems auditing. Throughout the remainder of this book, we assign this broader meaning to the audit risk model.

To apply the model, auditors first choose their level of desired audit risk. External auditors consider such factors as the level of reliance external parties are likely to place on the financial statements and the likelihood of the organization encountering financial difficulties subsequent to the audit. Internal auditors also consider these factors. In addition, they assess the short- and long-run consequences for their organizations if they fail to detect real or potential material losses from ineffective or inefficient operations.

Next auditors consider the level of inherent risk. Initially auditors consider general factors such as the nature of the organization (e.g., Is it a high flyer?), the industry in which it operates (e.g.. Is the industry subject to rapid change?), the characteristics of management (e.g.. Is management aggressive and autocratic?), and accounting and auditing concerns (e.g.. Are creative accounting practices used?). Auditors then consider the inherent risk associated with different segments of the audit (cycles, application systems, and financial statement accounts). For each segment, auditors consider such factors as the following:

Inherent Risk Factor	*Explanation*
Financial system	Those systems that usually provide financial control over the major assets of an organization—e.g., cash receipts and disbursements, payroll, accounts receivable and payable—often have higher inherent risk. They are frequently the target of fraud and embezzlement.
Strategic system	Those systems that provide an organization with a competitive advantage—e.g., systems that lock in customers or suppliers or embody a patent or trade secret—often have high inherent risk. They might be the target of industrial espionage or retaliatory actions by a competitor.
Critical operational systems	Those systems that could cripple an organization if they fail—e.g., customer reservations systems or production control systems—often have high inherent risk.
Technologically advanced systems	Those systems that use advanced technology often have high inherent risk because they are complex and the organization lacks experience with them.

To assess the level of control risk associated with a segment of the audit, auditors consider the reliability of both management and application controls. Auditors usually identify and evaluate controls in management subsystems first. Management (subsystem) controls are fundamental controls because they cover all application systems. Thus, the absence of a management control is a serious concern for auditors. Conceptually, management controls constitute protective layers of "onion skins" around applications (Figure 2-4). Forces that erode asset safeguarding, data integrity, system effectiveness, and system efficiency must penetrate each layer to undermine a lower layer. To the extent the outer layers of controls are intact; the inner layers of controls are more likely to be intact. In addition, it is often more efficient if auditors evaluate management controls before application controls. After auditors have evaluated a management control, auditors usually do not have to evaluate it again because it should function across all applications. For example, if auditors find that an organization enforces high-quality documentation standards, it is unlikely they will have to review the quality of documentation for each application system.

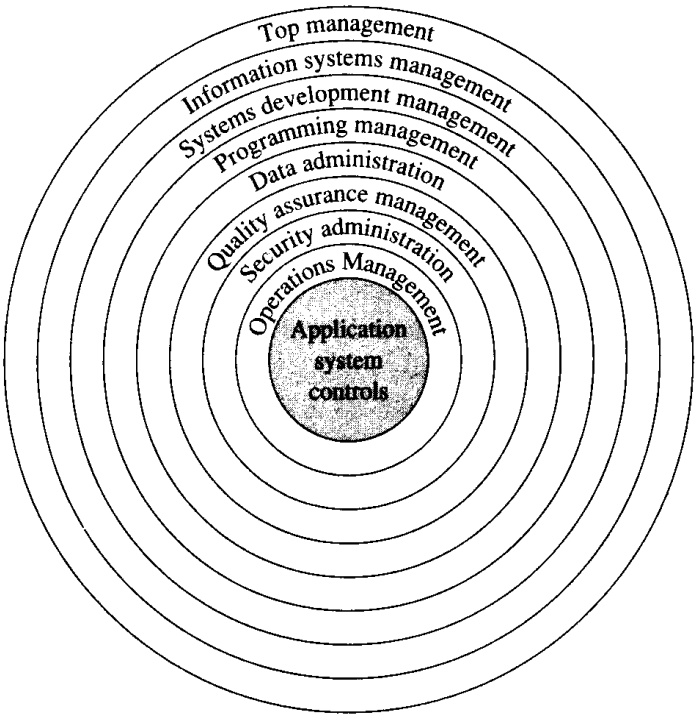


FIGURE 2-4. Management controls as an onion skin around application controls

Next auditors calculate the level of detection risk they must attain to achieve their desired audit risk. They then design evidence collection procedures in an attempt to achieve this level of detection risk, to estimate the level of detection risk they might achieve with a set of audit procedures, they must have a good understanding of how likely these audit procedures are to detect a material loss or account misstatement when one exists. Moreover, auditors must evaluate how reliably the audit procedures are likely to be applied. Not only must they choose audit procedures that have the capacity to provide us with a desired level of detection risk, they also must ensure they are properly executed.

In summary, the whole point to our considering the audit risk model is that audit efforts should be focused where they will have the highest payoffs. In most cases auditors cannot collect evidence to the extent they would like. Accordingly, they must be astute in terms of where they apply their audit procedures and how they interpret the evidence they collect. Throughout the audit they must continually make decisions on what to do next. Their notions of materiality and audit risk guide them in making this decision.

2.5 Types of audit procedures

When external auditors gather evidence to determine whether material losses have occurred or financial information has been materially misstated, they use five types of procedures:

1. Procedures to obtain an understanding of controls: Inquiries, inspections, and observations can be used to gain an understanding of what controls supposedly exist, how well they have been designed, and whether they have been placed in operation.

2. Tests of controls: Inquiries, inspections, observations, and reperformance of control procedures can be used to evaluate whether controls are operating effectively.

3. Substantive tests of details of transactions: These tests are designed to detect dollar errors or irregularities in transactions that would affect the financial statements. For example, an external auditor might verify that purchase and disbursement transactions are correctly recorded in journals and ledgers.

4. Substantive tests of details of account balances: These tests focus on the ending general ledger balances in the balance sheet and income statement. For example, an external auditor might circularize a sample of customers to test the existence and valuation of the debtors balance.

5. Analytical review procedures: These tests focus on relationships among data items with the objective of identifying areas that require further audit work. For example, an external auditor might examine the level of sales revenue across time to determine whether a material fluctuation that requires further investigation has occurred in the current year.

Auditors can use similar types of procedures if they are concerned wit evaluating the effectiveness and efficiency of an organization's operations:

1. Procedures to obtain an understanding of controls: Inquiries, inspections, and observations can be used to gain an understanding of the administrative controls set up to achieve effectiveness and efficiency objectives rather than the accounting controls set up to achieve asset safeguarding and data integrity objectives.

2. Tests of controls: Tests of controls focus on whether administrative controls have been well designed and whether they are operating effectively. For example, auditors might interview an operations manager to check whether she regularly reviews the response-time performance of a critical online system and, so, what action she takes when response times are unacceptable.

3. Substantive tests of details of transactions: From an effectiveness and efficiency perspective, auditors still have a notion of substantive tests of details of transactions. Using the response-time example discussed previously, auditors might check the response times for a sample of individual transactions to determine whether they are within acceptable bounds.

4. Substantive tests of overall results: The notion of account balances does not app in the context of effectiveness and efficiency concerns.

Nevertheless, auditors have a notion of overall effectiveness and efficiency results. For example, management might assert the average response time for an application system over 12-month period is two seconds. As a substantive test of this claimed overall re suit, auditors might survey users of the system to determine its validity.

5. Analytical review procedures: Analytical review procedures are still relevant the context of effectiveness and efficiency concerns. For example, auditors might build a queuing model or a simulation model of an application system evaluate whether the resources consumed by the application system seem reasonable.

Often, the order of tests from the least costly to the most costly is as fi lows: analytical review procedures, procedures to obtain understanding controls, tests of controls, substantive tests of details of transactions, and substantive tests of balances/overall results. On the other hand, the order is reversed when we consider the reliability and information content of the evidence provided by the different audit procedures. Accordingly, auditors usually carry out the less costly audit procedures first in the hope the evidence obtained from these procedures indicates it is unlikely a material loss or material misstatement has occurred or will occur. If this outcome arises, auditors can alter the nature, timing, and extent of the more costly tests used. For example, on the basis of their understanding ot controls and tests of controls, auditors might conclude controls are well designed and operating effectively. In this light, they would seek to reduce the costs of substantive testing in the following ways: change the nature of substantive testing by employing less costly substantive tests directed toward internal parties rather than external parties; change the timing of substantive testing by spreading it across a longer period to reduce costs; and change the extent of substantive tests by choosing smaller sample sizes to reduce costs.

2.6 Overview of steps in an audit

Keeping in mind the lessons in the previous sections on the nature of controls, the importance of system factoring in reducing complexity, the nature and consequences of audit risks, and the types of audit procedures auditors can carry out. Figure 2-5 flowcharts the major steps to be undertaken in an audit. The general approach shown in the flowchart is representative of the approaches advocated by many professional bodies of auditors.

The following subsections briefly describe each step in an audit and high-light those parts of the audit where the information systems auditor often plays an important role. Although Figure 2-5 and the ensuing discussion imply a sequential progression of audit steps, some steps can be carried out concurrently and some iteration of steps can occur. For example, some tests of controls could be carried out as auditors attempt to understand the controls that are supposed to be in place. Furthermore, while both external and internal auditors will follow the general approach shown in Figure 2-5,

the decisions they take at each step in the audit might vary because they have different roles. For example, internal auditors might spend more time testing controls because they are more concerned than external auditors about the efficiency of the controls. The following discussion points out how external and internal auditors might differ in the decisions they take at each stage of the audit.

2.6.1 Planning the Audit

Planning is the first phase of an audit. For an external auditor, this means investigating new and continuing clients to determine whether the audit engagement should be accepted, assigning appropriate staff to the audit, obtaining an engagement letter, obtaining background information on the client, understanding the client's legal obligations, and undertaking analytical review procedures to understand the client's business better and identify areas of risk in the audit. For an internal auditor, this means understanding the objectives to be accomplished in the audit, obtaining background information, assigning appropriate staff, and identifying areas of risk.

During the planning phase, auditors must decide on the preliminary materiality level to he set for the audit. An external auditor's concerns will he the size of misstatements in the financial statements that would affect the decisions of users of the financial statements. Internal auditors might also be concerned about the size of losses that have arisen or might arise through ineffective or inefficient operations.

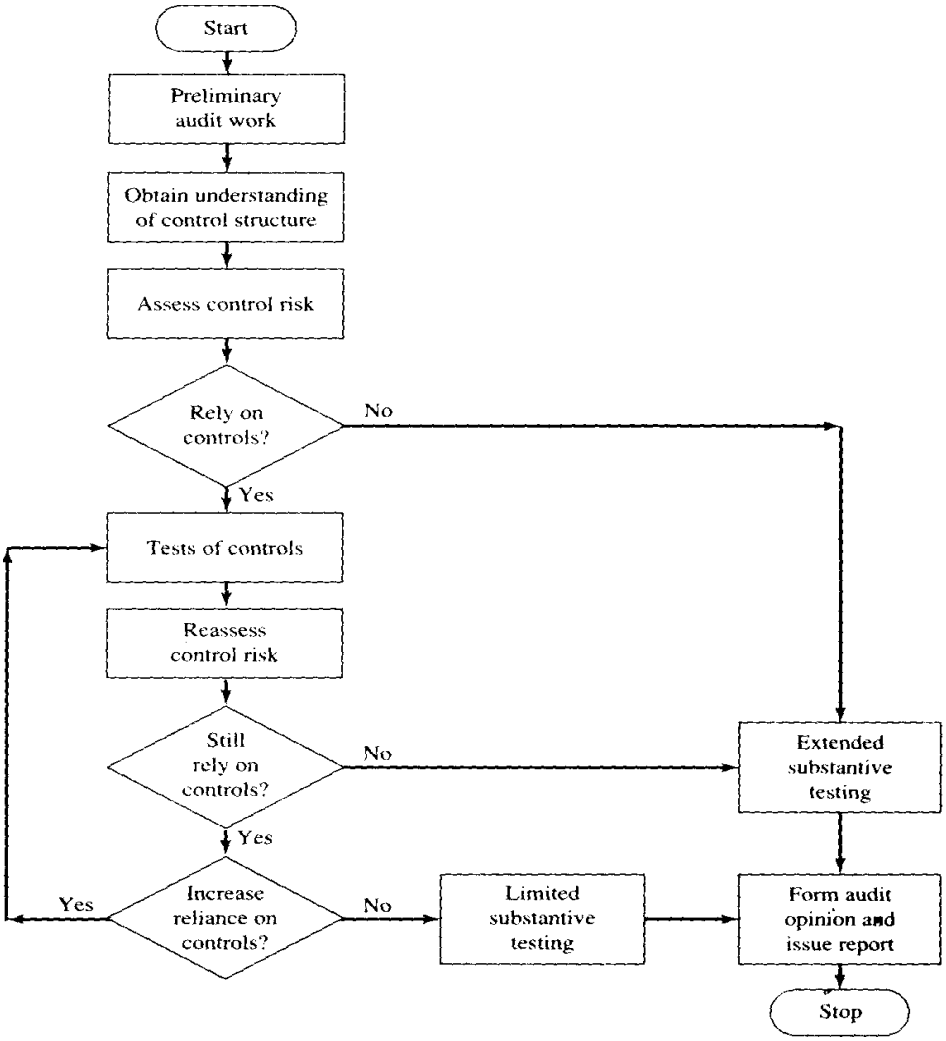


FIGURE 2-5. Flowchart of major steps in an IS audit

Auditors must also make a judgment on desired audit risk. Usually the level of desired audit risk is set for the overall audit rather than for segments of it. For external auditors, this reflects the risk they are willing to take to issue an unqualified opinion even though the financial statements are materially in error. For internal auditors, desired audit risk might also reflect the risk they are willing to take to issue an unqualified opinion even though material losses have occurred or might occur through ineffective or inefficient operations.

The levels of inherent risk will vary across different segments of the audit. Some segments are more susceptible to errors, irregularities, ineffectiveness, and inefficiencies. Auditors must consider each segment of the audit in turn and evaluate the factors that lead to inherent risk associated with the segment. For example, systems that involve handling of cash are susceptible to defalcations; technologically complex systems are susceptible to inefficient use of resources.

Perhaps the most difficult decision to make in the planning phase is the judgment on the level of control risk associated with each segment of the audit. When making this judgment, information systems audit skills are especially important. The American Institute of Certified Public Accountants argues that to decide on the level of control risk, auditors must first understand the internal controls used within an organization. Internal controls comprise five interrelated components:

1. Control environment: Elements that establish the control context in which specific accounting systems and control procedures must operate. The control environment is manifested in management's philosophy and operating style, the ways authority and responsibility are assigned, the way the audit committee functions, the methods used to plan and monitor performance, and so on.

2. Risk assessment: Elements that identify and analyze the risks faced by an organization and the ways these risks can be managed.

3. Control activities: Elements that operate to ensure transactions are authorized, duties are segregated, adequate documents and records are maintained, assets and records are safeguarded, and independent checks on performance and valuation of recorded amounts occur. These elements are usually called accounting controls. Internal auditors, however, also might be concerned with administrative controls established to achieve effectiveness and efficiency objectives.

4. Information and. communication: Elements in which information is identified, captured, and exchanged in a timely and appropriate form to allow personnel to discharge their responsibilities properly.

5. Monitoring: Elements that ensure internal controls operate reliably over time.

In the context of the concepts examined earlier in this chapter and the role information systems auditors perform, understanding internal controls in an organization involves factoring and examining both management controls and application system controls. Auditors can understand the control environment and risk assessment components primarily by examining management controls. For example, when auditors determine whether an information systems steering committee exists, they seek to understand the control environment and risk assessment components of internal control. Auditors can understand specific control activities by reviewing both management controls and application controls. For example, when auditors review those activities associated with production release of programs or entry of data to an application system, they are seeking to understand the control activities undertaken. Auditors can understand the information and communication component by examining both management controls and application controls. For example, when auditors examine how management communicates roles and responsibilities or how transactions are captured, recorded, processed, and summarized within an application system, they are seeking to understand the information and management component.

Auditors can understand the monitoring component primarily by examining management controls. For example, when auditors examine the ways management evaluates employee performance, they are seeking to understand the monitoring component.

Management controls can differ substantially from organization to organization. For example, an organization might have all information processing performed at a single site that is under the control of a single information systems department. In this situation, there is only one management system to evaluate that associated with the information systems department. Auditors would factor this system into various subsystems top management, systems development management, programming management, and so on and seek to understand internal controls in the context of each of these subsystems.

On the other hand, another organization's information systems function might be widely dispersed. For example, the organization might have a highly decentralized structure. Divisions might have responsibility for developing, operating, and maintaining their own information systems. Each might have its own computer center and information systems staff. End-user computing also could be substantial. Some end users might be developing, maintaining, and operating their own systems. In these circumstances, auditors must evaluate multiple management systems: one for each divisional site and perhaps one for each major end-user computing site. They must consider each management system in turn, evaluate the risks associated with each, and factor those that are material into their various subsystems. In short, auditors might have to examine multiple top-management subsystems, multiple systems development management subsystems, multiple programming management subsystems, and so on, to understand the internal controls.

Application controls also might be substantially diverse. In a highly centralized organization, there might be only one set of cycles to evaluate. In a highly decentralized organization, however, there might be multiple sets of cycles, each of which must be evaluated. For example, each division might have its own sales and collections cycle, payroll and personnel cycle, acquisitions and payments cycle, conversion, inventory, and warehousing cycle, and treasury cycle. Auditors must identify those cycles that are material to the audit, factor the cycles into application systems and subsystems, understand these systems and subsystems, and identify the controls that have been implemented over each important class of transactions that passes through the different systems and subsystems.

There are several types of evidence collection techniques used to understand the internal controls: review of working papers from prior audits, interviews with top management and information systems personnel, observations of activities carried out within the information systems function', and reviews of information systems documentation. The evidence can be documented by completing questionnaires, constructing high-level flowcharts and decision tables, and preparing narratives. A computer can be

helpful to employ these techniques and auditors might use a computer-aided software engineering (CASE) tool to draw flowcharts. Similarly, they might interact with questionnaire software that elicits responses on the status of various types of internal controls (see Chapter 16). Auditors must be careful not to undertake too much work, however, during this phase. The goal is to obtain just enough information to understand internal controls and to decide how to proceed with the audit.

After obtaining a satisfactory understanding of the internal controls, auditors must assess the level of control risk. External auditors assess control risk in terms of each major assertion that management should be prepared to make about material items in the financial statements (Table). Thus auditors must relate their understanding of internal controls to the impact they ultimately have on the figures presented in the financial statements. In the case of management controls, the relationship usually is indirect and if careful control is exercised over program maintenance, auditors might have increased confidence that a specific control in an application system will continue to be exercised properly throughout the financial period. As a result, they would be confident the control supports, say, the completeness assertion for a particular financial statement component. In the case of application controls, the relationship to financial statement components is usually fairly direct. If the control has not been designed properly or has not been operated effectively, the potential impact on a financial statement assertion is usually clear.

Assertion	*Explanation*
Existence	Assets, liabilities, and equities included in the financial statements actually exist.
Occurrence	All transactions represent events that have actually occurred.
Completeness	All transactions that have occurred have been recorded. All accounts have been presented in the financial statements.
Rights and obligations	Assets are the rights and liabilities are the obligations of the organization at the balance sheet date.
Valuation or allocation	Asset, liability, equity, revenue, and expense accounts have been recorded at the correct amounts.
Presentation and disclosure	Items in the financial statements are properly classified, described, and disclosed.

Internal auditors can also assess control risk in terms of assertions that management implicitly or explicitly make about the effectiveness and efficiency of systems. For example, management might say a system achieves a certain throughput rate and that customers of the organization who use the output of the system have a certain level of satisfaction with the performance of the system. Auditors must use their understanding of the internal controls to evaluate whether they have been designed appropriately and whether they have been placed in operation to support management's assertions.

After auditors obtain an understanding of the internal controls, they then must determine the control risk in relation to each assertion:

1. If auditors assess control risk at less than the maximum level, they must then identify the material controls that relate to the assertion and test the controls to evaluate whether they are operating effectively. They work on the assumption that tests will show that if the controls are operating effectively they can reduce the extent of substantive testing needed to reach an audit opinion.

2. If auditors assess control risk at the maximum level, they do not test controls; they might conclude that internal controls are unlikely to be effective and therefore cannot be relied upon or that a more effective and efficient audit can be conducted using a substantive approach.

2.6.2 Tests of Controls

Auditors test controls when they assess the control risk for an assertion at less than the maximum level. They rely on controls as a basis for reducing more costly testing. At this stage in the audit, however, auditors do not know whether the controls identified operate effectively. Tests of controls, therefore, evaluate whether specific, material controls are, in fact, reliable.

This phase usually begins by again focusing first on management controls. If testing shows that, contrary to expectations, management controls are not operating reliably, there might be little point to testing application controls. If auditors identify serious management-control weaknesses, they might have to issue an adverse opinion or undertake substantive tests of transactions and balances or overall results. Auditors conduct the evaluation iteratively for each management subsystem and each application subsystem that is material to the assertion.

To illustrate how to test management controls, assume that as auditors came to an understanding of internal controls, they are informed that senior management regularly undertakes information systems planning. To test whether this control is operating effectively, they might examine the minutes of meetings held by senior management to evaluate whether they conscientiously attend to planning on a regular basis. In addition, they might request a copy of the current information systems plan to evaluate its quality.

Assume that as auditors came to an understanding of the internal controls, they have identified standards covering program documentation. In light of their discussions with programmers, they believe programmers comply with these standards when they write programs. To test the control, auditors might identify a sample of programs they consider material to audit objectives. They could then examine the documentation for these programs to determine whether, in fact, documentation exists and whether it complies with the standards.

If auditors conclude that management controls are in place and working satisfactorily, they then would evaluate the reliability of application controls by tracing instances of material classes of transactions through each significant control exercised in the various application subsystems. For each transaction considered, auditors evaluate whether the control is operating effectively.

To illustrate how to test application controls, assume that as auditors came to an understanding of internal controls, they identified a control that required an accounts manager to check that her control clerk cleared all errors reported during an update run of a batch application system. Auditors might select a sample of update reports generated during the financial period and check for a signature to indicate that the accounts manager was regularly checking the work of the accounts clerk.

Assume that as auditors came to an understanding of internal controls, they identified a control that required a data-entry operator to enter customer orders only if customers had provided a signed order form. Auditors might select a sample of orders that had been entered throughout the financial period and check to see each entered order is supported by a signed, hard-copy order.

After auditors have completed tests of controls, they again assess control risk. In light of the test results, they might revise the preliminary assessment of control risk downwards or upwards. In other words, auditors might conclude that internal controls are stronger or weaker than initially anticipated. They might also conclude that it is worthwhile to perform more tests of controls with a view to further reducing the substantive testing. Perhaps internal controls are stronger than initially believed. Accordingly, auditors conclude control risk has decreased and seek further evidence to support this assessment.

During the controls-testing phase, internal auditors and external auditors might differ in their approaches to the audit. If internal auditors identify control weaknesses, they might expand their investigations to gain a better understanding of the nature of and implications of these weaknesses. Their objective might be to provide in-depth recommendations to rectify the control weaknesses. External auditors, on the other hand, will tend to cut short their investigations when they identify control weaknesses and proceed to undertake expanded substantive tests in light of the increased control risk they perceive.

2.6.3 Tests of Transactions

From an attest perspective, recall auditors use tests of transactions to evaluate whether erroneous or irregular processing of a transaction has led to a material misstatement of financial information. Typical attest tests of transactions include tracing journal entries to their source documents, examining price files for propriety, and testing computational accuracy. The computer is quite useful to perform these tests, and auditors might use

generalized audit software to check whether the interest paid on bank accounts has been calculated correctly.

From an operational perspective, auditors use tests of transactions to evaluate whether transactions or events have been handled effectively and efficiently. For example, to indicate a database system's effectiveness, auditors might examine a sample of queries recorded on a transaction log to evaluate whether the queries have been generated by a wide cross-section of users of the database system. To evaluate efficiency, auditors might examine the turnaround times for a sample of jobs submitted to an application system. Again, the computer can help to carry out these tests. For example, auditors may use generalized audit software to select a sample of database queries from a transaction log for evaluation.

In an attest audit, auditors conduct tests of transactions at interim dates in order to reduce the amount of substantive tests of balances to be done at financial year end and thus to reduce the overall costs of the audit. In an operational audit for effectiveness and efficiency purposes, auditors also use tests of transactions at interim dates in order to reduce the amount of substantive testing of overall results to be done near the reporting date. For example, if the response times for a sample of transactions that occur throughout the period under review are satisfactory, auditors can reduce the number of users surveyed near the report date to determine whether they consider response times to be satisfactory. To follow this strategy, auditors must know an operational audit is required well in advance of the reporting date.

If the results of tests of transactions indicate that material losses have occurred or might occur or that financial information is or might be materially misstated, substantive tests of balances or overall results will be expanded. Auditors can use expanded tests of balances or overall results to obtain a better estimate of the losses or misstatements that have occurred or might occur.

2.6.4 Tests of Balances or Overall Results

Auditors conduct tests of balances or overall results to obtain sufficient evidence for making a final judgment on the extent of losses or account misstatements that occur when the information systems function fails to safeguard assets, maintain data integrity, and achieve system effectiveness and efficiency. In general, tests of balances or overall results are the most expensive of the audit. Thus, auditors should design and execute these tests carefully.

To understand the approach in this phase, consider, first, the asset-safeguarding and data-integrity objectives. Some typical substantive tests of balances used are confirmation of receivables, physical counts of inventory, and recalculation of depreciation on fixed assets. Recall that if auditors believe controls are reliable on the basis of prior audit work, they will limit the number and scope of these tests because material losses or material account misstatements that have arisen through failure to safeguard assets

and maintain data integrity are not expected. On the other hand, if auditors believe controls are not reliable, they will need to expand the extent of substantive tests of balances to estimate better the size of the losses and account misstatements.

Consider, now, the system-effectiveness and system-efficiency objectives. The tests conducted to estimate losses from failure to achieve these objectives are less clear cut than those associated with asset safeguarding and data integrity objectives. For example, auditors might work with users of an application system to estimate the losses they believe have arisen because the system does not provide them with the output they require to make high-quality decisions. As another example, auditors might attempt to estimate the costs of inefficiencies that have occurred because failures in information systems planning have resulted in inappropriate hardware purchases. Again, the extent of the audit work performed depends on the auditor's prior assessment of the reliability of administrative controls.

Computer support is often required to undertake substantive tests of balances or overall results effectively and efficiently. For example, generalized audit software can be used to select and print confirmations; an expert system can be used to estimate the likely bad debts that will arise with receivables; a simulation package can be used to estimate how much throughput of work has been lost because a hardware/software platform has been 'poorly configured. Recall that the focus of the tests is to estimate the size of losses and account misstatements. The computer is a critical tool in these efforts.

As with the prior phases, the nature and conduct of the audit work during this phase can vary considerably, depending on the type of organization auditors are examining. At one extreme the audit could be a small organization that has a single, centralized information systems function. The audit work focuses on the losses and account misstatements that might have arisen from a limited number of sources. At the other extreme, the audit could be of a large, decentralized organization in which the information systems function is widely dispersed. The audit work must be extensive to take into account losses and misstatements that could- have arisen from a large number of sources.

2.6.5 Completion of the Audit

In the final phase of the audit, external auditors undertake several additional tests to bring the collection of evidence to a close. For example, they undertake reviews for subsequent events (events that occur subsequent to the financial statement date but that affect the information that should be reported in the financial statements) and contingent liabilities (potential liabilities that must be disclosed in the financial statements). They must then formulate an opinion about whether material losses or account misstatements have occurred and issue a report. The professional standards in many countries require one of four types of opinion are issued:

1. Disclaimer of opinion: On the basis of the audit work conducted, the auditor is unable to reach an opinion.

2. Adverse opinion: The auditor concludes that material losses have occurred or that the financial statements are materially misstated.

3. Qualified opinion: The auditor concludes that losses have occurred or that the financial statements are misstated but that the amounts are not material.

4. Unqualified opinion: The auditor believes that no material losses or account misstatements have occurred.

In addition to asset safeguarding and data integrity concerns, internal auditors might also have to decide whether material losses have occurred because the information systems function has failed to achieve system effectiveness and efficiency objectives. Unlike the asset safeguarding and data integrity objectives, the form of the audit opinion relating to system effectiveness and efficiency objectives is not prescribed by professional standards. Therefore, auditors must formulate their wording for the opinion so that it clearly communicates the findings and judgment. Nevertheless, a typical report would include an introduction that describes the audit objectives, scope, and general approach employed, a summary of critical findings, recommendations to address the major issues that arise from the findings, and data to support the critical findings listed in the report.

Auditors are also concerned with prognoses about losses and account misstatements. In other words, even though auditors might have concluded no material losses or misstatements have occurred, they might believe control weaknesses exist that mean such losses or misstatements could occur in the future. These weaknesses might motivate a concern about the viability of the organization if a major threat eventuates. In addition, auditors might be concerned about contingent liabilities associated with losses that arise through significant control weaknesses. For example, customers could sue an organization if it cannot provide products or services because its computer systems are not operational. At the conclusion of an audit, therefore, an important function that auditors perform is to provide management with a report documenting any control weaknesses they have identified, the potential consequences of these control weaknesses, and some recommendations for remedial actions.

2.7 Auditing around or through the computer

When auditors come to the controls testing phase of an information systems audit, one of the major decisions they must make is whether to test controls by auditing around or through the computer. The phrases "auditing around the computer" and "auditing through the computer" are carryovers from the past. They arose during the period when auditors were debating how much technical knowledge was required to audit computer systems. Some argued that little knowledge was needed because auditors could evaluate computer

systems simply by checking their input and output. Others contended audits could not be conducted properly unless the internal workings of computer systems were examined and evaluated. Unfortunately, the arguments of the former group were sometimes motivated by their lack of technical knowledge about computers. Thus, among some auditors the phrase "auditing around the computer" had derogatory connotations. Today we recognize that the two approaches each have their merits and limitations and that each must be considered carefully in the context of planning and executing the most cost-effective audit.

2.7.1 Auditing Around the Computer

Auditing around the computer involves arriving at an audit opinion through examining and evaluating management controls and then input and output only for application systems. Based on the quality of an application system's input and output, auditors infer the quality of the application system's processing. The application system's processing is not examined directly. Instead, auditors view the computer as a black box.

Auditors should audit around the computer when it is the most cost-effective way to undertake the audit. This circumstance often arises when an application system has three characteristics. First, the system is simple and batch oriented. Sometimes batch computer systems are a straightforward extension of manual systems. They have the following properties:

1. Their inherent risk is low. They are unlikely to be subject to material errors or irregularities or to be associated with significant ineffectiveness or inefficiencies in operations.

2. Their logic is straightforward. No special routines have been developed to allow the computer to process data.

3. Input transactions are batched, and control is maintained using traditional methods for example, separation of duties and management supervision.

4. Processing primarily consists of sorting the input data and updating the master file sequentially.

5. A clear audit trail exists. Detailed reports are prepared at key points within the system.

6. The task environment is relatively constant and the system is rarely modified.

Second, often it is cost-effective to audit around the computer when an application system uses a generalized package as its software platform. If the package has been provided by a reputable vendor, has received widespread use, and appears error free, auditors might decide not to test the processing aspects of the system directly. Instead they might seek to ensure (1) the organization has not modified the package in any way; (2) adequate controls exist over the source code, object code, and

documentation to prevent unauthorized modification of the package; and (3) high-quality controls exist over input to and output from the package.

Note, however, that not all generalized software packages make application systems amenable to auditing around the computer. Some packages provide a set of generalized functions that still must be selected and combined to accomplish application-system purposes. For example, database management system software might provide generalized update functions, but a high-level program still must be written to combine these functions in the required ways. In this situation, auditors are less able to infer the quality of processing from simply examining the system's input and output.

Third, auditors might audit around the computer when a high reliance is placed on user rather than computer controls to safeguard assets, maintain data integrity, and attain effectiveness and efficiency objectives. In testing, the focus is on the reliability of user controls rather than the reliability of computer controls.

Usually auditing around the computer is a simple approach to the conduct of the audit, and it can be performed by auditors who have little technical knowledge of computers. The audit should be managed, however, by someone who has expertise in information systems auditing.

The approach has two major limitations. First, the type of computer system in which it is applicable is very restricted. It should not be used when systems are complex. Otherwise, auditors might fail to understand some aspect of a system that could have a significant effect on the audit approach. Second, it does not provide information about the system's ability to cope with change. Systems can be designed and programs can be written in certain ways to inhibit their degradation when user requirements change. For internal auditors, the system's robustness could be an important concern in light of their effectiveness and efficiency objectives.

2.7.2 Auditing Through the Computer

For the most part, auditors are now involved in auditing through the computer. They use the computer to test (1) the processing logic and controls existing within the system and (2) the records produced by the system. Depending on the complexity of the application system, the task of auditing through the computer might be fairly simple, or it might require extensive technical competence on the part of the auditor.

Auditing through the computer must be used in the following cases:

1. The inherent risk associated with the application system is high.
2. The application system processes large volumes of input and produces large volumes of output that make extensive, direct examination of the validity of input and output difficult to undertake.

3. Significant parts of the internal control system are embodied in the computer system. For example, in an online banking system, a computer program might batch transactions for individual tellers to provide control totals for reconciliation at the end of the day's processing.

4. The processing logic embedded within the application system is complex. Moreover, large portions of system code are intended to facilitate use of the system or efficient processing.

5. Because of cost-benefit considerations, substantial gaps in the visible audit trail are common in the system.

The primary advantage of auditing through the computer is that auditors have increased power to test an application system effectively. They can expand the range and capability of tests they can perform and thus increase their confidence in the reliability of the evidence collection and evaluation. Furthermore, by directly examining the processing logic embedded within an application system, auditors are better able to assess the system's ability to cope with change and the likelihood of losses or account misstatements arising in the future.

The approach has two disadvantages. First, it can sometimes be costly, especially in terms of the labor hours that must be expended to understand the internal workings of an application system. Second, in some cases we will need extensive technical expertise if we are to understand how the system works. These disadvantages are really spurious, however, if auditing through the computer is the only viable method of carrying out the audit.

3. SUMMARY

Information systems auditing is an organizational function that evaluates asset safeguarding, data integrity, system effectiveness, and system efficiency in computer-based information systems. It has arisen for seven major reasons:

1. The consequences of losing the data resource;
2. The possibility of misallocating resources because of decisions based on incorrect data or decision rules;
3. The possibility of computer abuse if computer systems are not controlled;
4. The high value of computer hardware, software, and personnel;
5. The high costs of computer error;
6. The need to maintain the privacy of individual persons; and
7. The need to control the evolutionary use of computers.

Asset safeguarding, data integrity, system effectiveness, and system efficiency can be achieved only if a sound system of internal control exists. Use

of computers does not affect the basic objectives of internal control; however, it affects how these objectives must be achieved.

The use of computers affects both the evidence collection and evidence evaluation functions auditors perform. Computer control technology is often more complex than manual system control technology; consequently, it is more difficult to understand controls and collect evidence on the reliability of controls. Similarly, it is more difficult to understand the implications of a control strength or weakness for the overall reliability of a system.

Information systems auditing borrows much of its theory and methodologies from other areas. Traditional auditing contributes knowledge of internal control practices and an overall control philosophy. Information systems management provides methodologies necessary to achieve successful design and implementation of systems. Behavioral science indicates when and why information systems are likely to fail because of people problems. Computer science contributes knowledge about how hardware and software should be designed to be effective and efficient and to safeguard assets and maintain data integrity.

There are five major steps to be undertaken during the conduct of an audit. First, auditors must plan the audit. In particular, auditors must reach an overall understanding of internal controls. Second, if auditors expect to rely on internal controls, controls must be tested to evaluate whether they are operating effectively. Third, auditors must carry out substantive tests of details of transactions to evaluate whether a material loss or account misstatement has occurred or might occur. Fourth, auditors must carry out substantive tests of balances or overall results to gather sufficient evidence to make a judgment on the size of the losses or account misstatements that have occurred or might occur. Fifth, based on their evaluation of the evidence collected, auditors issue an audit opinion.

4. QUESTIONS

1. Why is there a need for control and audit of computer systems?
2. What are the implications of a company losing its:
   a. Personnel master file b. Inventory master file
3. How can inadequate controls in a computer system lead to incorrect decision making?
4. Why is the control stills needed to protect hardware, software, and personnel even though substantial insurance coverage might have been taken out by organization?
5. Why does the computer cause us to have increased concerns about the privacy of individuals?
6. What are the major assets in an information systems facility?
7. Define data integrity. What factors affect the importance of data integrity an organization?

8. How does the continuing evolution of computer hardware and software technology affect an auditor's ability to (a) understand controls, and (b) collect evidence on the reliability of controls?
9. What impact does the use of computers have on the nature and conduct of the evidence evaluation function carried out by auditors?
10. Briefly explain the contribution of the following areas to information systems.
11. Define the concept of a control.
12. Why must auditors focus on controls as a system?
13. Briefly explain the cycles approach to conducting an information systems audit.
14. Briefly describe two types of evidence collection procedures you might use to obtain an understanding of internal controls.
15. How do controls reduce expected losses?

5. REFERENCE BOOKS

1. Weber R; Information Systems Control and Audit (Person Education)
2. Dube: Information systems for Auditing (TMH)
3. Auditing Information Systems, 2nd Edition. Jack J. Champlain (Wiley)

UNIT – II

1. Programming Management Controls

Structure

Objectives

After going through this lesson, you should be able to know:

- how to implement program development life cycle
- how to Organizing the programming team
- how to Managing the system programming group

1.1 Introduction

In this lesson we examine those practices that lead to the production or acquisition of high-quality software. We begin by examining the major phases in the program development life cycle. The discussion highlights the types of good practices that should exist and the control concerns that auditors have with respect to each phase. Next we examine alternative ways of organizing and managing programming teams. In particular, from a control perspective we focus on the advantages and disadvantages of the different team structures that can be used. Finally, we examine the special control problems that arise in relation to the activities of system programmers. We consider some approaches that can be used to alleviate these control problems.

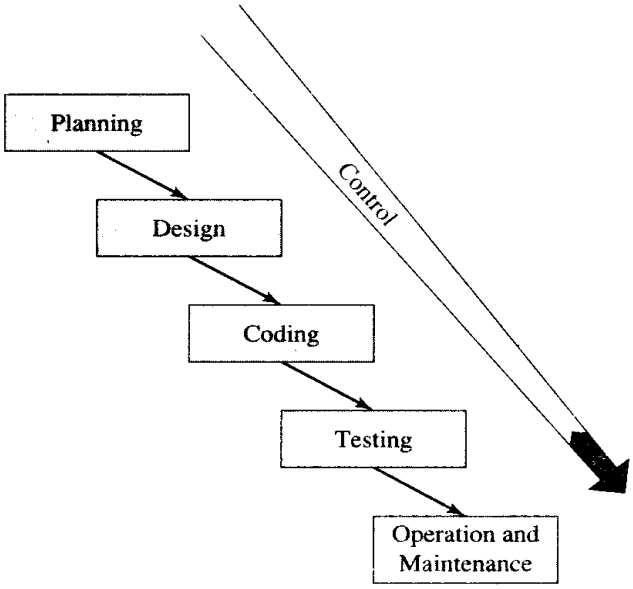
1.2 The program development life cycle

As program development and acquisition is a major phase within the systems development life cycle. The primary objectives of this phase are to produce or acquire and to implement high-quality programs. Some major characteristics of high-quality programs follow:

1. They perform their functions correctly and completely.
2. They have a high-quality user interface.
3. They work efficiently.
4. They are well designed and well documented.
5. They are easy to maintain.
6. They are robust under abnormal conditions.

If programs are to have these characteristics, development, acquisition, and implementation activities must be well managed. As with systems development, auditors can use a life-cycle model to better understand, plan, and carry out the tasks that must be undertaken to obtain high quality software. During audits, this model can also guide the conduct of their evidence-collection and evidence-evaluation activities.

The following sections provide normative guidelines for six major phases in the program development life cycle (Figure 1-1): (1) planning, (2) control, (3) design, (4) coding, (5) testing, and (6) operation and maintenance. As with the systems development life cycle, the conduct of these phases can vary considerably, depending on certain contingencies. We examine the effects of such contingencies on each phase and the ways in which the audit approach must be adjusted to take them into account.



FIGURES 1-1 Program development life cycle

1.2.1 Planning

Perhaps the major task that management must undertake during the planning phase is to estimate the amount of resources (especially worker hours) required for software development, acquisition, and implementation. If, for example, the software is to be written in house and the development and implementation task is substantial, management might attempt to estimate the lines of source code to be written or the number of function points to be produced. These estimates might then be extrapolated to the number of worker hours required to produce the software.

1. Estimate the number of domain items for the program.
2. Assign a weight to each domain item based on experience and expert advice.
3. Compute F = Sum (F_i) using the following table.
4. Compute a complexity adjustment C based on such factors as whether data communications or distributed processing functions are required.
5. Compute the function point value, FP, using the following formula:

FP =F* (0.65 + 0.01 C) where the constants 0.65 and 0.01 are determined empirically.

If software is to be developed and implemented in house, management should use the five major software cost-estimation techniques identified by Boehm:

1. Algorithmic models: These models estimate resources needed based on a set of "cost drivers" for example, the estimated number of source instructions to be written, the programming language to be used, and the

volatility of the requirements definition. A well-known example is Boehm's COCOMO model.

2. Expert judgment: Experts might estimate the resources needed to undertake the programming project.

3. Analogy: If a similar software project has been undertaken already, resource requirements can be estimated based on this prior experience.

4. Top-down estimation: The project is first subdivided into its various tasks, and resource requirements for each task are then estimated.

5. Bottom-up estimation: If the tasks to be undertaken are fairly well defined at the outset, the resource needs for each can be estimated and aggregated to obtain those needed for the entire project.

If software is to be acquired, three major types of costs that will be incurred are purchase costs, the costs associated with contracting for and implementing the software, and the costs of operating and maintaining the software. Potential vendors will provide estimates of purchase costs and operation and maintenance costs. Expert judgment, analogy, top-down estimation, and bottom-up estimation might still be used, however, to predict the costs associated with contracting for and implementing the software.

Besides estimating resource requirements, management must address several other important decisions during the planning phase:

Decision	*Explanation*
Design approach	If the program is to be developed in house, management must choose the design approach to be used, e.g., prototyping or some type of top-down or bottom-up approach.
Implementation approach	The software could be written in house, contractors could be employed, or a package could be purchased. If the software must be coded, decisions must be made on the programming language to use and the coding discipline to be applied.
Integration and testing approach	Testing might require special resources, e.g., simulators or special hardware to monitor performance. Responsibility for integration and testing must be assigned.
Project team organization	If a project team must be formed to develop or acquire the software, the membership and structure of the team must be determined.

The importance and complexity of planning decisions can vary considerably, depending on such factors as the size of the software to be developed or purchased and the uncertainty relating to user requirements or support technology. For example, consider the extent of resource planning needed for a large project involving thousands of lines of programming code versus a small project that can be programmed using a spreadsheet package. Similarly, consider the extent of test and integration planning needed for a large, extensively modified, purchased software package versus a small, off-the-shelf, unaltered, purchased software package. Clearly, the approaches to and effort expended on planning across these cases should not be the same.

Auditors should have two major concerns about the conduct of the planning phase. First, they should evaluate whether the nature of and extent of planning are appropriate to the different types of software that are developed or acquired. They can gather this evidence in the normal ways for example, interviews, observations, and reviews of documentation. Second, they must evaluate how well the planning work is being undertaken. For example, they might assess how accurately resource requirements have been estimated. If they conclude planning is well done, they should be more confident about the conduct of the remaining phases in the software life cycle. If planning is problematic, however, the conduct of the remaining phases may be undermined.

How difficult it will be to carry out evidence collection and evaluation will vary, depending on the extent to which responsibility for software development, acquisition, and implementation is dispersed throughout the organization. If, for example, end users employ high-level languages to develop programs that are material in the context of audit objectives, auditors must evaluate the quality of the planning work end users are performing, wherever they are located. The audit work will have to extend beyond the boundaries of the information systems department. If, however, responsibility for software development and acquisition is confined to a single, central group, the evidence collection and evaluation tasks are easier.

1.2.2 Control

The control phase has two major purposes. First, task progress in the various software life-cycle phases should be monitored against plan. Any significant deviations detected form the basis for corrective action. Second, control over software development, acquisition, and implementation tasks should be exercised to ensure software released for production use is authentic, accurate, and complete. Note in Figure 1-1 that the control phase is a "phantom" phase that extends in parallel with all other phases of the software life cycle.

To help monitor progress against plan, several techniques can be used. Work breakdown structures (WBS) can be prepared to identify the specific tasks that have to be undertaken to develop, acquire, and implement

software (Figure 1-2). Detailed resource requirements then can be estimated for each task. These estimates form the basis for monitoring progress.

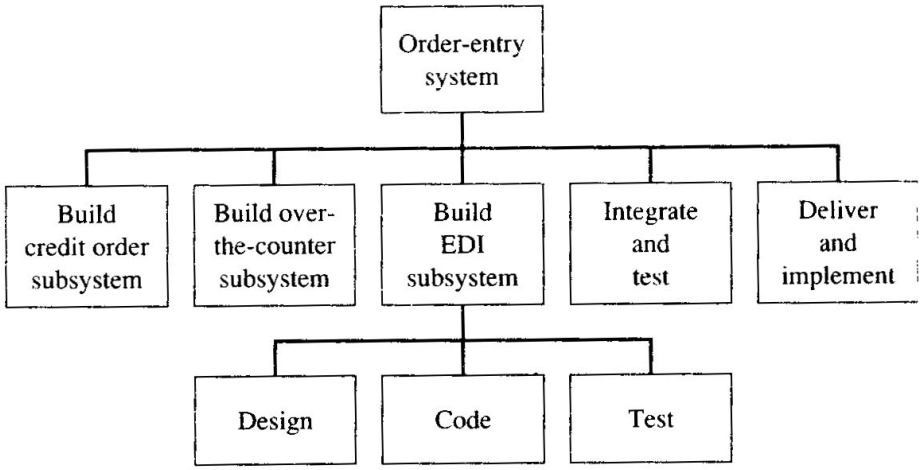


FIGURE 1-2 Partial work breakdown structure for an order-entry system

Gantt charts can be prepared to help schedule tasks (Figure 1-3). They show when tasks should begin and end, what tasks can be undertaken concurrently, and what tasks must proceed serially. They help identify the consequences of early or late completion of a task. The actual progress of a software project can be plotted on a Gantt chart to show pictorially whether the project is on track.

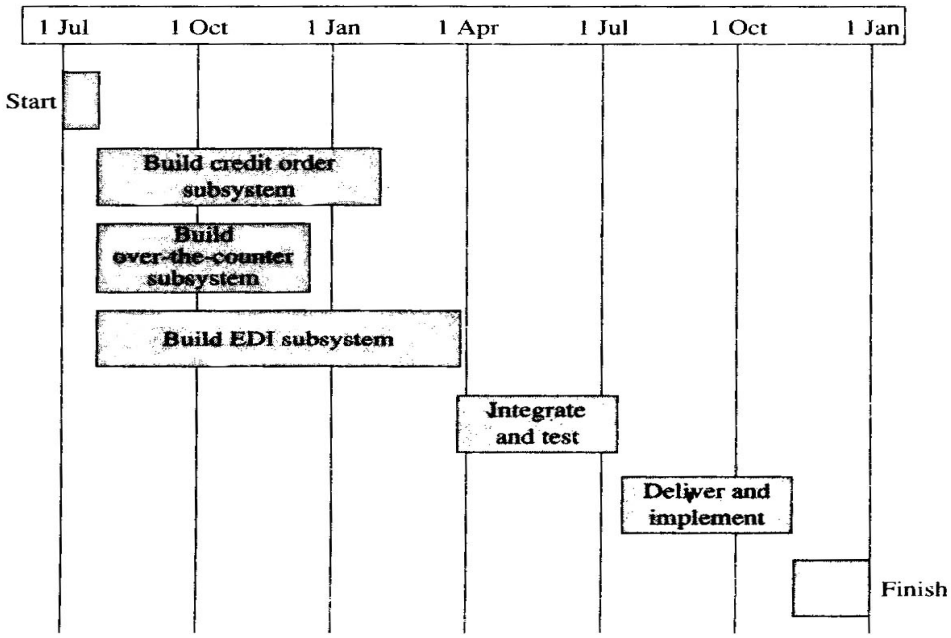


FIGURE 1-3 Gantt chart for order-entry system

Program evaluation and review technique (PERT) charts show the tasks that have to be undertaken, how they are interrelated, and the resource requirements for each task (Figure 1-4). They allow the critical path to be determined that is, the path along which any delay in completion of a task will result in the overall software project being delayed (the bold line in Figure 1-4). Like Gantt charts, therefore, they enable management to determine the consequences of early or late completion of a task.

To help ensure that authentic, accurate, and complete software is released, management must establish review procedures and access controls. Review procedures can be undertaken as each major milestone during software development, acquisition, or implementation is reached. The quality of work per formed up to that point should be assessed, and a decision must be made on whether the project should proceed to the next phase. In some cases, a formal review process might be undertaken. For example, in a large software-development project that will have a widespread impact, a review team might be constituted comprising representatives of all the major stakeholder groups. This team is responsible for evaluating the quality of work completed and approving subsequent work. In other cases, the review is somewhat informal. For example, end users might be developing software using fourth-generation languages or acquiring off-the-shelf software that is material from an audit perspective. Nevertheless, the software might have only a localized impact on the organization. In this situation, less onerous review and approval procedures could provide a satisfactory level of control.

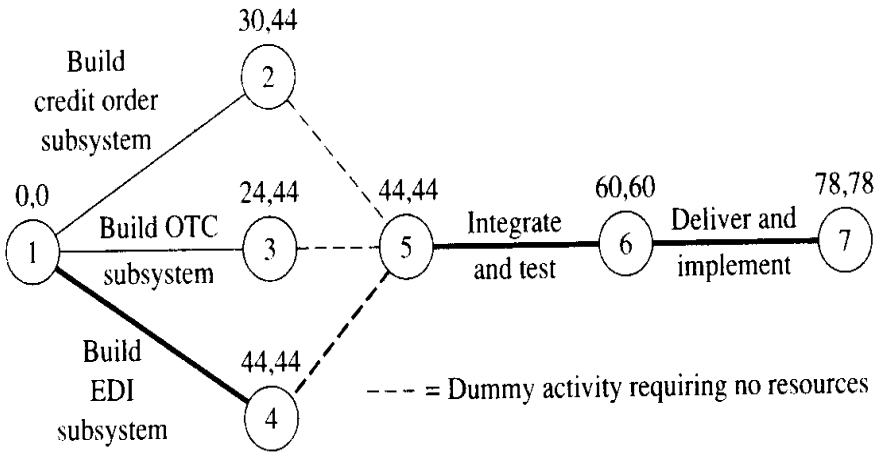


FIGURE 1-4 PERT chart for order-entry system

Both manual and automated access controls can be established over the development, acquisition, and maintenance of software. Manual controls can be used, for example, to restrict access to hard-copy documentation. Without management approval, a librarian may not permit a programmer to remove program documentation from a library.

Automated controls can be established via program library software. This software permits source and object code files to be established. Access to

these files can then be controlled via passwords. The software provides an audit trail of accesses to and changes to source and object files that management can review for propriety. In some cases, program library software can provide a means of ensuring compliance with licensing agreements pertaining to acquired software. For example, access controls in the software reduce the threat of software piracy.

Many organizations also use library software to set up separate test and production libraries of source and object files. Programmers are not permitted to access the production library. When a developed or acquired program has been approved for production release, a separate control group transfers the program from the test library to the production library. If a program must be maintained, the control group transfers a copy of the program to the test library, where it can be accessed by the programmer authorized to maintain it.

As with the planning phase, the nature of and importance of procedures exercised in the control phase can vary considerably, depending upon the type of software being developed, acquired, or maintained. Consider, for example, the control procedures that should be used in the development of a large electronic data interchange program versus a small spreadsheet application. As the materiality of software increases, clearly control procedures become more critical. As materiality decreases, however, less rigorous control procedures can be used.

Auditors should have two concerns about the conduct of the control phase. First, they must evaluate whether the nature of and extent of control activities undertaken are appropriate for the different types of software that are developed or acquired. Auditors should identify those locations in an organization where material software is being produced or purchased and determine whether the control procedures in place are appropriate for the varying levels of materiality of the software.

Second, auditors must gather evidence on whether the control procedures are operating reliably. For example, they might first choose a sample of past and current software development and acquisition projects carried out at different locations in the organization they are auditing. They might then use observations, interviews, and a review of documentation to determine whether management regularly monitors progress against plan. If the organization audited uses program library software, auditors might also choose a sample of programs and examine the audit trail of accesses to and maintenance of these programs to evaluate whether unauthorized activities have occurred.

1.2.3 Design

If programs are to be developed or acquired software is to be modified, design activities must be undertaken. During the design phase, programmers seek to specify the structure and operation of programs that will meet the requirements articulated during the information processing system design phase of systems development. In small systems, a single

program might be able to meet these requirements. For example, one spreadsheet program might satisfy the requirements specified for a decision support system application. Larger systems, however, might have been broken up into various job steps. Different programs must be designed to satisfy the requirements associated with each job step.

During the design phase, the auditor's primary concern will be to find out whether programmers use some type of systematic approach to design. The auditor must vary expectations depending on such factors as the size and materiality of the program. The need for rigorous, systematic design increases as programs become larger and more material. Auditors must also vary expectations depending on the type of personnel who develop programs. They are likely to have fewer concerns if a centralized, professional information systems group develops programs relative to, say, dispersed, and end-user groups.

To some extent, design approaches will also vary depending on the type of programming language that has been or will be used to implement the program. For example, if programmers believe they can develop a program using spreadsheet software, they will think about satisfying requirements in the context of matrices and operations on matrices. Auditors should see design practices that evidence the kinds of ideas shown in Table 1-1.

If, on the other hand, programmers use a third-generation language (such as COBOL), the auditor should check to see whether some type of structured design approach has been used. The relative merits of these different structured design approaches are still a contentious matter, but auditors usually need not be concerned about the debate. Providing at least some type of structured design approach is used, they can have increased confidence in the quality of work performed during the design phase.

If programmers use an object-oriented programming language, they will probably use some type of object-oriented approach to program design. Again, although object-oriented design approaches might vary, an auditor's primary concern will be to determine whether at least one of the more widely accepted approaches is used.

Systematic design is still important even when fourth-generation languages are used to implement programs. Fourth-generation languages alleviate many of the detailed design issues that must be addressed with third-generation languages for example, how files will be opened, closed, and accessed. Nevertheless, poor-quality fourth-generation programs will be produced if programmers do not give sufficient thought to the structure and dynamics of the programs they are writing. Design approaches like functional decomposition, data flow design, and data structure design are still useful in a fourth-generation language environment.

Auditors can obtain evidence of the design practices used by undertaking interviews, observations, and reviews of documentation. They can talk with programmers to determine whether they have an understanding of the need for systematic design approaches and, if so, whether and how

they use them. Auditors can observe programmers at their work to determine whether they are using systematic approaches to program design. They can review program documentation to determine whether it contains items like structure charts as evidence those programmers are using a systematic approach to design.

TABLE 1-2 Some Good Spreadsheet Design Practices

1. Start with a design plan.
2. Separate data-entry areas from calculation areas.
3. Store constants and parameters in a separate area.
4. Design data-capture areas to mirror existing data-capture forms.
5. Enter data in either rows or columns but not both.
6. Place instructions and meaningful names in the spreadsheet itself.
7. Protect critical cells/formulas with the cell-protect feature.
8. To speed up data entry, use the manual recalculation feature for large spread sheets.
9. Use range names for related cells whenever possible.

1.2.4 Coding

The coding phase is undertaken when software is to be developed or acquired software is to be modified. During the coding phase, programmers write and document source code in some programming language to implement the program design. Sometimes coding can proceed concurrently with the design and testing phases. Some part of a program is designed; it is then implemented and tested.

Programming management must attend to several major issues during the coding phase. Each is discussed briefly in the following subsections, together with some audit implications.

1.2.5 Module Implementation and Integration Strategy

As programs become larger, management must give more consideration to the order in which modules will be coded. Management must also decide how individual modules will eventually be integrated.

Three major module implementation and integration strategies that can be used follow:

Strategy	*Explanation*
Top down	Using this strategy, high-level modules are coded, tested, and integrated before low-level modules. An advantage of this strategy is that errors in high-level module interfaces, which are sometimes the most serious types of errors, are identified early. A disadvantage is that program testing might be difficult when low-level modules perform critical input-output functions.
Bottom up	Using this strategy, lower-level modules are coded, tested, and integrated before higher-level modules. An advantage of this strategy is that low-level modules that are critical to the eventual operation of the program are implemented and tested first. In some cases, modules that have been developed for other applications might be reused or modified. A disadvantage of the approach is that it is difficult to observe the overall operation of the program until late in its implementation.
Threads	Using this strategy, a decision is first made on the order in which program functions should be implemented. The modules that support each function are then determined, and each set is then implemented in decreasing order of functional importance. An advantage of this strategy is that the most important functions are implemented first. A disadvantage is that subsequent integration of modules might be more difficult, compared with a top-down or bottom-up approach.

Auditors should seek evidence on the level of care exercised by programming management in choosing a module implementation and integration strategy. Especially with large programs, use of a poor strategy can seriously undermine the quality of the program produced. Auditors might use interviews, for example, to determine whether management employs a systematic approach to choosing a module implementation and integration strategy. They might also examine program documentation to obtain evidence on the types of strategies that have been adopted.

1.2.6 Coding Strategy

Irrespective of the module implementation and integration strategy chosen, programming management must ensure that program code is written whenever possible, according to structured programming conventions. These conventions constrain code to three basic control structures, none of which require a "GO TO" mechanism:

1. Simple sequence (SEQUENCE);
2. Selection based on a test (IF-THEN-ELSE); and
3. Conditional repetition (DO-WHILE).

In addition, each module in a program should have only one entry and one exit point, the length of modules should be restricted to about 50-100 source statements, and a top-down flow of control should be used.

If structured programming conventions are followed, it is generally believed that programmers write source code that contains fewer errors, is easier to

understand, and is easier to maintain. This belief holds irrespective of the type of programming language used. Programs written in a fourth-generation language like SQL, for example, benefit in the same ways as programs written in a third-generation language like COBOL. Even when high-level code must be written in, say, a macro for a spreadsheet application, the code is more likely to be error-free, easier to understand, and easier to maintain if structured programming conventions are followed.

Auditors should seek evidence to determine whether programming management ensures that programmers follow structured programming conventions. They can interview managers and programmers and ask them about the practices they follow, observe programmers at their work, and examine program documentation to determine how code has been written. If high-quality code has been produced, auditors can have increased confidence that programs meet their objectives. If low-quality code has been written, however, auditors might conclude that they have to expand the extent of substantive test procedures.

Auditors should also check to see whether programmers employ automated facilities to assist them with their coding work. Some useful types of automated coding facilities follow:

If auditors find that programmers are using automated facilities, they can have more confidence in the quality of their coding work. Automated facilities reduce the likelihood of human errors and irregularities, improve programmer productivity, and enhance greater standardization of work.

1.2.7 Documentation Strategy

High-quality source code documentation is an important means of reducing coding errors when a program is initially written and facilitating subsequent maintenance of the program. Some generally accepted guidelines for improving the quality of program documentation follow:

1. Provide charts that show the overall makeup of the program in terms of its major components and the relationships among these components.

2. Use comment lines liberally throughout a program to explain the nature of the program, its various components, and the flow of logic. Program header comments can be used to describe the overall purpose of the program, major functions performed by the program, and important files used. Module comments can be used to explain the function performed by a module and how it fits into the overall program. Line comments can be used to elucidate complex pieces of logic in the program.

3. Use names for variables, constants, types, paragraphs, modules, and sections that are meaningful to the readers of program source code. Meaningful names can greatly enhance the self-documenting features of a program.

4. Lay out program source code so it is easy to read. For example, each sentence should begin on a new line; subsequent lines belonging to a sentence should be indented; statements following conditional tests should be indented; white space should be used to set off related blocks of code.

5. Group related types of code together. For example, in a spreadsheet program, data-entry areas can be separated from areas where computations are performed. In a third-generation language, all the variables and constants relating to a particular module or function can be grouped together.

Several automated tools are available to assist programmers to produce well documented program code. For example, editors can be used to ensure that data items are named consistently and pretty-print facilities can be used to make program code more readable. To the extent these types of automated tools are used, higher-quality documentation is likely to be produced.

Perhaps the quickest way for auditors to determine the quality of program documentation is to obtain a sample and to examine it for evidence of high quality practices. If the documentation quality is low, auditors should have concerns about the care with which software is being developed. In addition, auditors should have concerns about how well subsequent modifications to and maintenance of software can be undertaken. If the quality of documentation is high, they can place increased reliance on the software and decrease the extent of their substantive test procedures.

1.2.8 Testing

During the testing phase, a developed or acquired program is evaluated to determine whether it achieves its specified requirements. Testing can identify program design errors or program coding errors. In some cases, testing may also pinpoint inaccurate or incomplete specifications. Note, however, that testing can identify only the presence of errors. It cannot identify their absence.

Unless tests are designed specifically to tease out an error, most likely the error will remain undetected.

As with the previous phases of the program development life cycle, the testing phase can vary considerably, depending on such factors as the size of the program and its materiality. Nevertheless, testing often involves seven steps:

1. Select the boundaries of the test: Testing can focus on an individual module in a program, several modules, or the entire program.
2. Determine the goals of the test: Testing can be used to identify unauthorized, inaccurate, incomplete, ineffective, or inefficient code. A particular test should focus on only one (or a small number) of goals for example, the performance of the program under load stress.

3. Choose the testing approach: Several testing approaches have been developed and are now widely used for example, black-box testing and white-box testing. These approaches are discussed in subsequent sections.
4. Develop the test: Test data or test scenarios must be developed that will accomplish the goals of the test. In particular, the expected results of the test must be determined.
5. Conduct the test: The conduct of the test can involve, for example, executing test data through a program or performing a hand-simulation of the program's execution pattern under various test scenarios.
6. Evaluate the test results: The actual results obtained under the test must be compared against the expected results. The nature of any discrepancies identified must be determined.
7. Document the test: All steps in the testing process must be documented.

Auditors usually pay special attention to the testing phase of the program development life cycle. The quality of testing can have a major impact on how well other phases are performed. High-quality testing forces programmers to undertake other phases of the program development life cycle rigorously. Furthermore, in many software projects, experience has shown that testing consumes 40 percent to 70 percent of development and implementation resources. Given audit concerns with effectiveness and efficiency, therefore, testing is often a material item auditors must consider during system development.

Testing because it is an important means auditors use to gather evidence on the quality of a system they are evaluating. The following sections provide an overview; however, of three levels of testing that can be conducted during program testing: unit testing, integration testing, and whole-of-program were testing. They describe the nature of each level of testing, how it might be undertaken, and some concerns auditors should have. They also describe how the conduct of each level of testing might vary depending upon various contingencies, such as the size of the program and whether the program has been developed or acquired.

1.2.9 Unit Testing

Unit testing focuses on evaluating individual modules within a program. Thus, unit testing tends to be undertaken only for large programs in which individual modules constitute substantive pieces of work. If a program is being developed in house or under contract, the applicability of unit testing should always be considered. If an off-the-shelf program has been purchased, however, unit testing is unlikely to be employed unless the program has been modified in some way.

Two major types of unit tests may be undertaken. The first type, static analysis tests, evaluates the quality of a module through a direct examination of source code. The module is not executed on a machine,

although it might be executed in someone's mind. Some important types of static analysis checks follow:

Type of Test	*Explanation*
Desk checking	Desk checking involves a programmer examining the module's code for evidence of errors or irregularities. For example, the programmer might look for syntax errors, logic errors, deviations from coding standards, or fraudulent code. Desk checking can be performed by the programmer who coded the module or by someone else.
Structured walk-throughs	In a structured walk-through, the programmer responsible for the design and coding of a module leads other programmers through the module with the aim of detecting errors or irregularities. The review group comprises independent programmers, although they might be on the same project team as the programmer who coded the module. The product of the review process is a list of defects to be corrected and not a list detailing how they are to be corrected.
Design and code inspections	In a design and code inspection, a special review team is appointed to review a program module's code. A trained moderator leads the review team through the review. Each review team member must follow defined participant rules. Formal checklists are used to guide the review, and the results are documented using preprinted forms. Follow-up procedures are used to ensure all errors or irregularities identified are rectified. Overall, the procedures used during a design and code inspection are more formal than those used during a structured walk-through.

Some types of static analysis can be undertaken using automated tools. For example, these tools can identify deviations from coding standards, variables that are declared but never initialized or used, module calling sequences, and parameters passed between modules.

The second type of unit testing that can be undertaken is a dynamic analysis test. Unlike static tests, dynamic tests require the module to be executed on a machine. Two important types of dynamic tests follow:

Type of Test	*Explanation*
Black-box test	In a black-box test, the internal logic of a module is not examined (Figure 1-5a). Instead, test cases are designed based on the requirements specification for the module. The test cases are then executed to determine deviations from requirements. Black-box testing might not reveal functions performed by a module that are excessive to the requirements specification for the module.
White-box test	In a white-box test, test cases are designed after the internal logic of a module has been examined (Figure 1-5b). The test cases are meant to traverse the different execution paths built into the program. Although a white-box test reveals the internal workings of a module, it might not identify requirements that the module fails to satisfy.

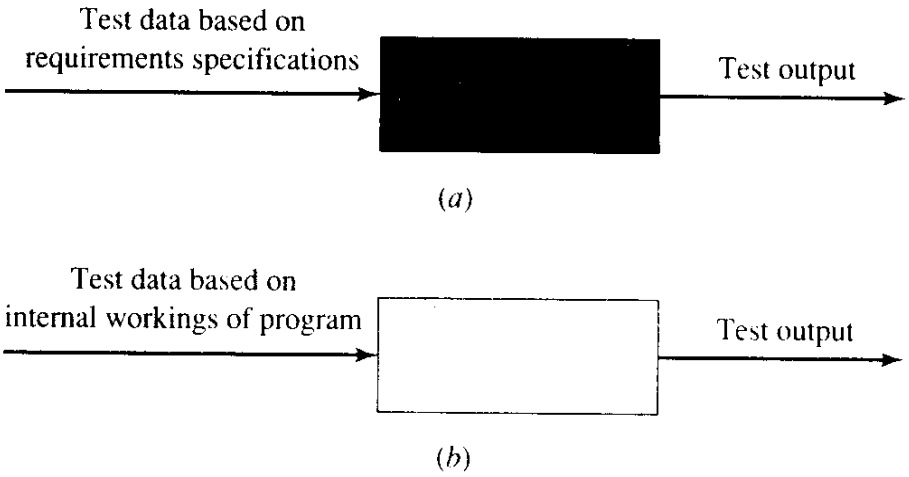


FIGURE 1.5(a) Black-box testing of programs, (b) White-box testing of programs.

As with static analysis, automated tools are also available to assist programmers undertake dynamic analysis of a module. For example, tools are available to generate test cases, control the execution of the test, vary the workload under which the module must perform, capture output, identify which execution paths have been traversed by test data, and compare actual output with expected output.

Auditors can use interviews, observations, and examination of documentation to evaluate how well unit testing is conducted. They can ask programmers how they undertake unit testing, observe them at their work, and choose a sample of test documentation to evaluate how well static analysis and dynamic analysis are performed. They should see evidence of automated tools being used to improve the quality of unit testing.

Unless end users are developing large programs, most likely audit concerns about unit testing will focus on the work done by information systems profes-

sionals. This latter group typically has responsibility for developing the sorts of programs where unit testing is needed. Furthermore, they typically have responsibility for evaluating acquired software where unit testing is an important facet of the overall contractual process.

1.2.10 Integration Testing

Integration testing focuses on evaluating groups of program modules primarily to determine (1) whether their interfaces are defective, and (2) overall, whether they fail to meet their requirements specifications. In some cases, integration testing can also be used to determine whether the performance of a set of modules degenerates under high workloads and whether processing is carried out efficiently.

Like unit testing, integration testing is typically undertaken only when larger and more complex programs are being developed either in house or via contractors. Auditors, therefore, will most likely be concerned primarily with the quality of integration testing work carried out by information systems professionals rather than end users. The amount of integration-testing work carried out by end users is likely to be small.

Two different strategies can be used to undertake integration testing: big-bang testing and incremental testing (Figure 1-6). When big-bang testing is used, all individual modules are coded, tested individually, and then assembled in total to perform the integration tests. In short, all intermodule dependencies are tested together, and in effect integration testing disappears. The test really becomes a whole-of-program test. Big-bang testing might be an efficient way to proceed in small to moderate-size systems, but it is risky if complex intermodule dependencies exist. Critical errors or irregularities may not be identified on a timely basis.

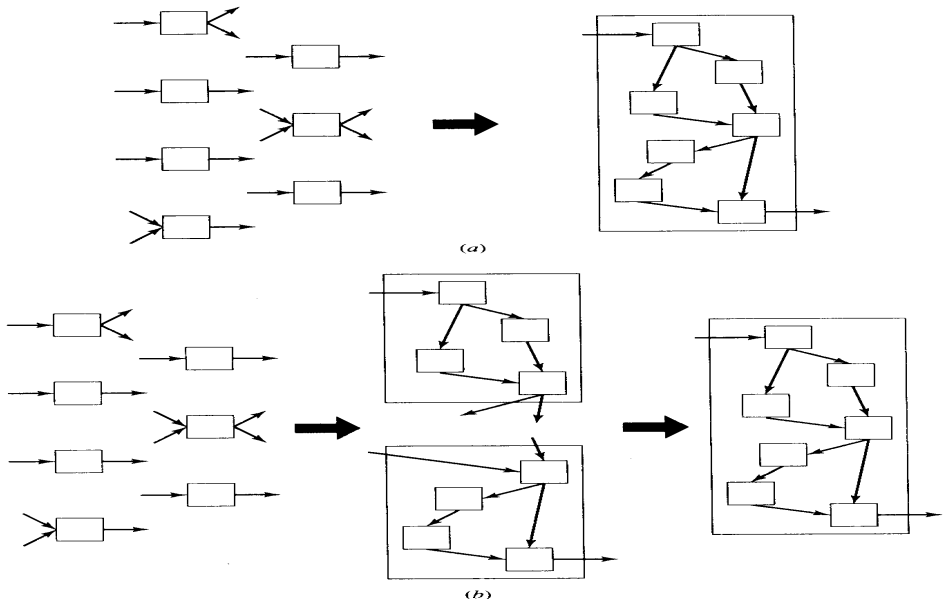


FIGURE 1-6 (a) Big-bang testing (b) Incremental testing.

Incremental testing means subsets of modules are assembled iteratively and tested until the total program is finally in place. Depending on the design and coding strategy used to develop the program, three types of integration-testing approaches may be used:

Type of Test	*Explanation*
Top-down test	In a top-down integration testing approach, the top-level modules are tested first. Lower-level modules that are not yet implemented are simulated via *stubs*, which are dummy modules that simply confirm the interface is working correctly.
Bottom-up test	In a bottom-up testing approach, the bottom-level modules are tested first. Higher-level modules that are not yet implemented are simulated via *drivers*, which are dummy modules that simply confirm the interface is working correctly.
Hybrid test	A hybrid integration testing approach involves a combination of top-down and bottom-up integration testing. This approach is sometimes called *sandwich testing*.

An auditor's primary concern will be to see that a systematic approach to integration testing has been chosen for those programs where it is applicable and that the approach chosen has been well executed. Auditors can gather evidence via interviews, observations, and reviews of documentation for a sample of programs. They should also check whether automated tools have been used to enhance the quality of integration testing.

1.2.11 Whole-of-Program Testing

Whole-of-program tests focus on the program, in total, to determine whether it meets its requirements. Even when a program is too small to make unit testing or integration testing worthwhile, whole-of-program testing should be undertaken to evaluate its quality. Similarly, if unit testing or integration testing is inappropriate because a program has been purchased, whole-of-program testing should still be undertaken When programs are acquired, whole-of-program tests are an important means of determining whether developers have fulfilled their part of the contract.

Pfleeger identifies four types of whole-of-program tests that might be undertaken:

Type of Test	*Explanation*
Function test	Function tests are used to determine whether the integrated program fulfills its requirements.
Performance test	Performance tests are used to determine whether the integrated program achieves certain performance criteria, some of which might not be formally specified in the requirements: fault tolerance, reliability under load, maintenance of security, response time adequacy, throughput rate adequacy.
Acceptance test	Function tests and performance tests are performed by the programmers responsible for developing the program. Acceptance tests are performed by the end users of a program. Acceptance tests also focus on determining whether the program fulfills its requirements specifications and any general performance requirements.
Installation test	Whereas function tests, performance tests, and acceptance tests can be performed in a test environment, installation tests are performed in the operational environment. For example, the program is executed on the machine that will be used for operational purposes rather than a test machine.

An auditor's primary concerns will be to see that whole-of-program tests have been undertaken for all material programs and that these tests have been well-designed and executed. Whereas unit testing and integration testing might not always be applicable during program development or acquisition, whole-of-program testing should always be undertaken. Auditors can use observations, interviews, and reviews of documentation to assess how well it has been done, expecting that evidence-gathering activities will be dispersed more widely with whole-of-program testing compared with unit and integration testing. Whereas the latter two types of testing primarily will be carried out by information systems professionals, whole-of-program testing could be undertaken by anyone in the organization who has responsibility for the development, implementation, or acquisition of programs. Whole-of-program testing should be undertaken by end users, for example, who develop material programs using high-level programming languages.

1.2.12 Operation and Maintenance

A program becomes operational when it is released for day-to-day use within an organization. From an audit perspective, our primary concern with the operational use of a program is that its performance be monitored properly. Someone must be responsible for identifying when a program needs to be maintained. Otherwise, timely identification of maintenance needs might not occur. As a result, the program might corrupt a database, fail to meet user requirements, or operate inefficiently. Formal mechanisms for monitoring the status of operational programs are especially important when the users of a program are dispersed widely throughout an organization.

As programs carry out their day-to-day work, three types of maintenance might be needed to keep them operational: (1) repair maintenance errors might be discovered that have to be corrected; (2) adaptive maintenance user needs may change and the program has to be altered accordingly; and (3) perfective maintenance the program could be tuned to decrease its resource consumption. Overall, these three types of maintenance tasks often account for a substantial part of the cost of owning software. Some estimates place maintenance costs, on average, at two-thirds of the total cost of owning a program.

1.3 Organizing the programming team

The way in which programmers are organized to undertake their work can have an important effect on the quality of the resulting software and the resources consumed to produce the software. They can be organized functionally, whereby each performs a specialized activity for example, COBOL coding, maintenance work, or communications programming. Functional structures work best when projects are straightforward and they can be decomposed into relatively self-contained, small- to moderate-sized tasks. They also allow programmers to develop specialist skills and, therefore, can reduce the resource consumption required to complete a task.

Alternatively, programmers can be organized as teams. They work for some period of time on a project. At the conclusion of the project, the team might be disbanded. Teams facilitate communication among their members. Accordingly, they are useful organizational structures when the project to be undertaken is complex and uncertain. On the other hand, team structures incur overheads associated with the high levels of interaction that can occur among their members and the overall management of the team.

Much software development, acquisition, and implementation is now undertaken by teams. In this light, the following subsections examine three major team structures that are used to organize programmers. Each has its advantages and disadvantages. From an audit perspective, the concern is to see that management has chosen a team structure carefully in light of the project size, the level of uncertainty/complexity facing the project, and the level of slack that exists in the project schedule. Moreover, auditors should be concerned to see that management has chosen the team members carefully so their skills and temperament are suited to the team structure in which they must work.

1.3.1 Chief Programmer Teams

In 1971, IBM completed a project for The New York Times newspaper. The system developed and implemented was an online retrieval system for the newspaper's file of clippings'. Throughout the project, IBM used a programming team structure that was radically different from the typical ways programming teams were then organized. For the size of the project about 83,000 lines of source code the results were impressive: The project

was delivered on time with few errors; programmer productivity was very high; errors discovered were easy to correct.

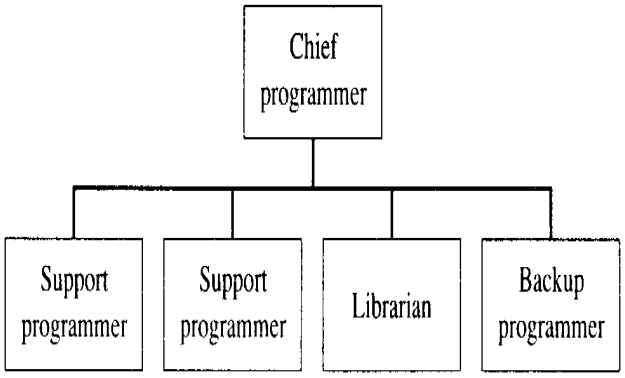


FIGURE 1-7 Chief programmer team organization structure.

The particular organization structure used by IBM on The New York Times project is known as a chief programmer team. A chief programmer team is simply a specific form of a project-based organizational structure one that has a high level of centralized control. Figure 1-7 shows the structure of a chief programmer team. The functions of the various personnel who are members of the team follow:

Team Member	*Functions*
Chief programmer	Ultimately responsible for the system on which the team works; must be an expert, highly productive programmer; responsible for designing, coding, and integrating the critical parts of the system; assigns work to the backup and support programmers.
Backup programmer	A senior programmer responsible for providing full support to the chief programmer; must be capable of assuming the chief programmers' duties at any time.
Support programmer	Needed for large projects that cannot be handled by the chief programmer and backup programmer alone; provides specialist support; assists in coding and testing lower-level modules.
Librarian	Responsible for maintaining the program production library (discussed subsequently); submits input and collects output for programmers; files output from compilations and tests; keeps source-code and object-code libraries up to date.

The chief programmer team structure is designed to reduce the need for information processing among the team members and to increase their capacities to process information. It achieves these objectives in three ways. First, it reduces the number of communications channels needed among team members by minimizing the number of personnel on the team. As a

consequence, however, it places more onerous productivity requirements on each team member to compensate for the loss of worker resources.

Second, each member of the chief programmer team performs specialized tasks. The chief programmer primarily is responsible for designing, coding, and testing the system. The backup programmer and support programmers provide specialized support; for example, they might advise the chief programmer on the intricacies of the operating system. The librarian relieves the chief programmer, backup programmer, and support programmers of the routine, clerical duties associated with the system. Thus, the structure aims at improving productivity by having team members do what they do best.

TABLE 1-2 Choosing a Programming Team Structure

Type of Team	*Level of Uncertainty*	*Level of Complexity*	*Size of Task*	*Time Constraints*
Chief programmer team	Low-moderate	Low-moderate	Moderate	Tight
Adaptive team	Moderate-high	Moderate-high	Moderate	Loose
Controlled-decentralized team	Moderate-high	Moderate-high	Large	Moderate

Third, the team's capacity to process information is increased by having the librarian perform a lateral, coordinating role. Central to this role is a program production library consisting of two parts: internal and external. The internal part comprises source code, object code, linkage commands, job control statements, and so on. It is maintained solely by the librarian, not the programmers. The external part comprises folders containing compilation results, test results, and other supporting documentation. The programmers work only with the external library, making whatever changes they need on program listings or coding sheets. The librarian then implements these changes. Each team member has access to the external library; thus, code, test results, etc., are public. Programmers are encouraged to examine each other's work so errors or potential interface problems are identified.

Chief programmer teams will be most successful when the task is well defined and moderate in size (Table 1-2). Centralized structures like a chief programmer team inhibit the information flows that are needed to generate innovative alternatives when the task is uncertain. Nevertheless, by controlling interaction among group members, chief programmer teams are more likely to meet tight deadlines than decentralized groups. For large tasks, the productivity requirements placed on members of a chief programmer team become too onerous. Other types of team structures that allow more members need to be adopted.

1.3.2 Adaptive Teams

Weinberg proposes another type of team structure for programmers: an adaptive team (Figure 1-8). Like chief programmer teams, adaptive teams comprise only a small number of persons. The structure of the team is meant to cater for two sets of needs: (1) the organization's requirements for quality programs to be produced and (2) the social/psychological needs of each programmer in the team.

Adaptive teams differ from chief programmer teams in three ways. First adaptive teams have no hierarchy of authority. The leadership of the team rotates among its members. The person having greatest skill with the activity undertaken usually assumes the leadership for the duration of that activity. Second, in an adaptive team, tasks are assigned to members of the team rather than defined positions. In the assignment of tasks, the objective is to exploit the strengths and avoid the weaknesses of each team member. Thus, there is no notion of a chief programmer with a defined role, a backup programmer with a defined role, and so on; an adaptive team is a self-organizing entity. Third, an adaptive team has no formal librarian role to perform a lateral coordinating function. Instead, team members are responsible for carefully examining and evaluating one another's work. The intent is to foster a feeling of joint responsibility for the quality of the programming product. At the same time, team members cannot have an ego attachment to the work they perform if open evaluation is to exist; hence, this type of team is sometimes called an "egoless" team.

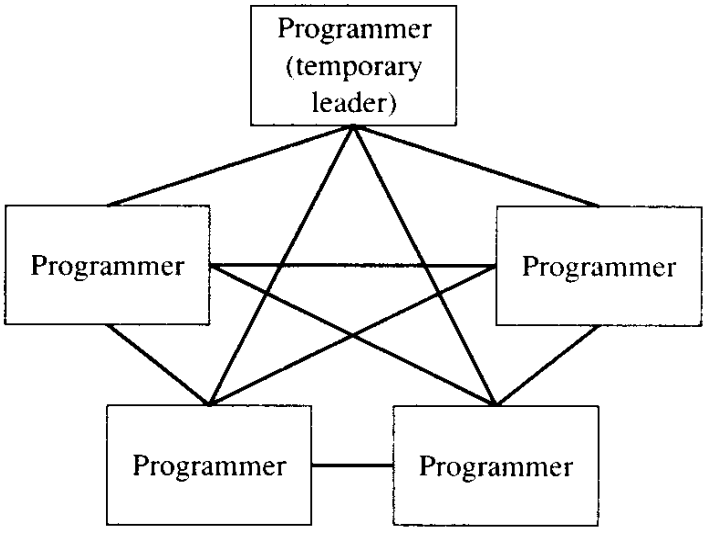


FIGURE 1-8 Adaptive team organization

The structure of an adaptive team is based upon the fact that substantial individual differences exist among programmers in their abilities to perform various types of programming tasks. It is also structured to allow the free

flow of information among team members. Thus, adaptive teams are suited to programming tasks where a high level of uncertainty exists (Table 1-2).

Nevertheless, Mantei points out they have several limitations. First, because an adaptive team is a form of decentralized organization structure, it will generate more communications than a centralized team. Although this increase is an advantage in a long-term, difficult project, it is a disadvantage when a project is subject to tight time constraints. Second, groups engage in riskier behavior than individual people because the effects of failure can be dispersed: thus, an adaptive team might adopt risky solutions to a programming problem. Third, contrary to expectations, adaptive teams sometimes discourage innovative programming solutions. Decentralized groups tend to exhibit greater conformity than centralized groups because they enforce uniformity of behavior and punish deviations from the norm.

To overcome some of the problems of adaptive teams, Constantine advocates using a structured open team for software engineering projects. A structured open team fosters collaborative teamwork and consensus to achieve both innovation and coordination. The team leader adopts a supportive and democratic style with team members, but nevertheless they have full external responsibility for the performance of the team.

1.3.3 Controlled-Decentralized Teams

A fourth type of organization structure for programmers is a controlled-decentralized team (Figure 1-9). This structure has a group of junior programmers reporting to senior programmers who in turn report to a project leader. Information flows occur within a group and upward through the senior programmer to the project leader. Thus, the controlled-decentralized team ideally reaps the benefits of both the chief programmer and adaptive team structures.

Mantei argues controlled-decentralized teams are best used when the programming task is large and difficult (Table 1-2). Large tasks cannot he accomplished by chief programmer teams, whatever the productivity of the team members. Moreover, complex problems are best solved by decentralized groups, and the group structure of the controlled-decentralized team facilitates problem solving. Nevertheless, controlled-decentralized teams do not work well when the programming task cannot be subdivided, nor are they suited to projects that must meet tight deadlines.

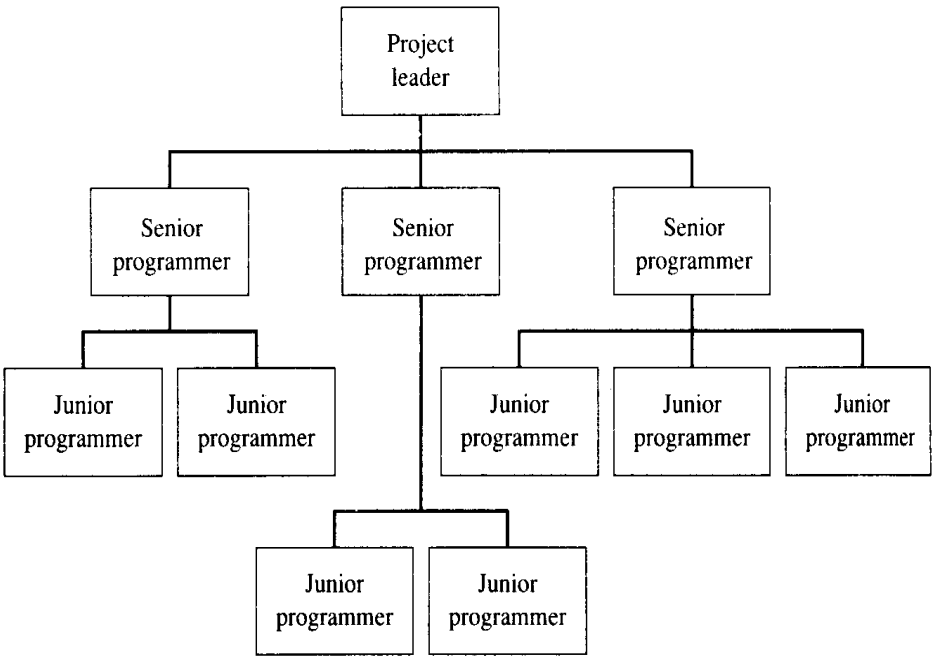


FIGURE 1-9 Controlled-decentralized organization structure.

1.4 Managing the system programming group

Programmers are often classified as either application programmers or system programmers. The former develop and maintain programs for application systems. The latter develop and maintain system software that is, software, such as operating systems, database management systems, and communications software, which provides general functions useful to a wide range of application software.

1.4.1 Control Problems

Both the nature of system software and the nature of system programming activities can present substantial control problems for management. System software is a critical, shared resource; thus, errors or irregularities in system software can affect any application systems that use it. Furthermore, system software often must operate in privileged mode to perform its functions; that is, it has a special execution status that enables it to circumvent many standard controls. This privileged status can be abused. For example, system software might be used to gain unauthorized access to private data that can be sold to competitors or to allow jobs to execute without being charged for resource consumption via the normal job accounting software. In the latter case, system programmers could be carrying out private consulting activities, for example, and using the machine as a free resource.

Controlling system programmers is a difficult task. They are usually highly skilled persons who often work alone or in small groups. Thus, exercising

traditional controls over their activities, such as separation of duties and independent checks on performance, is difficult. Moreover, they often work in crisis situations in which the need to get a job running overrides the need to maintain established control procedures. For example, the communications software might crash during a peak load period. A system programmer might be required to devise a quick "fix" so terminals can be reactivated and customers once again can be serviced.

1.4.2 Control Measures

In many organizations, management has tended to regard the system programming group as uncontrollable. Indeed, some information systems professionals argue the imposition of controls over system programmers will cause their work to deteriorate: They are sensitive, creative, often erratic persons who do not take kindly to restrictions. They need autonomy if they are to produce their best work, especially if they are subject to tight deadlines.

Auditors, however, should be skeptical of-these claims. Organizations that hold to these beliefs run the risk of major losses occurring through system programming errors or irregularities. Well-controlled system programming groups do exist. These groups experience neither high staff turnover nor poor quality work. Although it is difficult to exercise strong and varied controls over system programmers, some of the following measures can be implemented:

1. Hire only high-quality system programming staff. Compared with application programmers, management might undertake more in-depth background checking and interviewing when hiring system programmers.
2. Separate duties to the extent possible. If more than one system programmer is employed, duties should be separated to the extent possible. For example, responsibilities for designing and coding a system program might be separated from responsibilities for testing the program.
3. Develop and document methods and performance standards. System programmers should know what is expected of them in terms of how they perform their jobs. They should not be left to devise their own approaches, which could run contrary to the organization's control objectives.
4. Restrict the powers of system programmers. System programmers should not be allowed to "tinker" with the system software during production time. Moreover, they should be permitted to develop and test system software that runs in privileged mode only during special test periods. During production periods, system programmers usually should have only the same powers as application programmers.
5. Keep a manual and machine log of system programmer activities. Independent, secure logs of system programmer activities should be

kept. Periodically, these logs should be scrutinized to determine whether unauthorized activities have occurred.

6. Employ outside consultants to evaluate system programming work. If internal expertise is not available to evaluate the work of system programmers, outside experts might be hired from time to time to review the work of system programmers.
7. Have application programmers periodically evaluate system programmers. Although application programmers might not be capable of writing high-quality system software, they might still be able to evaluate the quality of work performed by the system programming group.

Even with these control measures in place and operating reliably, however, ultimately the best control might be to indoctrinate system programmers in the organization's policies. If system programmers see management exercising high ethical behavior and communicating a clear expectation that all employees must follow this norm, they will find it more difficult to rationalize any abuse of their powers.

Auditors should pay special attention to system programmers because of the high exposure associated with their activities. They should see at least some of the control measures described in this chapter in place and operating reliably. To the extent that controls over system programmers are weak, however, the consequences potentially are widespread. As discussed previously, errors or irregularities in system software can undermine the integrity of all application software that uses the system software. In this light, substantial reliance might have to be placed on compensating controls or the extent of substantive testing conducted may have to be expanded.

2. Security Management Controls

Structure

Objectives

After going through this lesson, you should be able to:

- understand how to Conducting a security program;
- discuss about Major security threats and remedial measures
- understand how Controls of last resort

2.1 Introduction

Information systems security administrators are responsible for ensuring that information systems assets are secure. Assets are secure when the expected losses that will occur from threats eventuating over some time period are at an acceptable level. Note three important aspects of this definition of security. First, we accept that some losses will inevitably occur. Eliminating all possible losses is either impossible or too costly. Second, we specify some level of acceptable losses. This level will dictate how much we are willing to spend on controls. Third, we must choose a time period. We determine what level of loss we would be willing to bear during this time period.

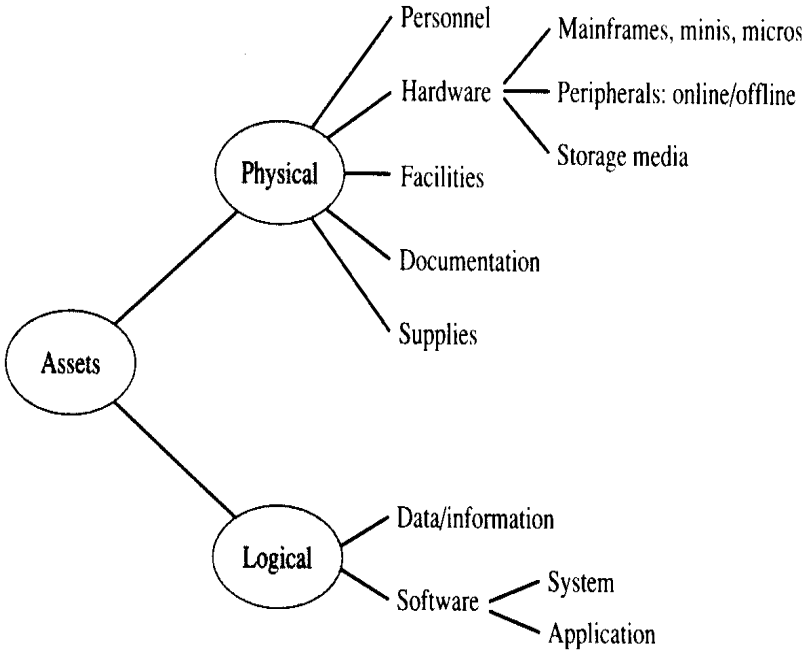


FIGURE 2-1 Categories of information systems assets.

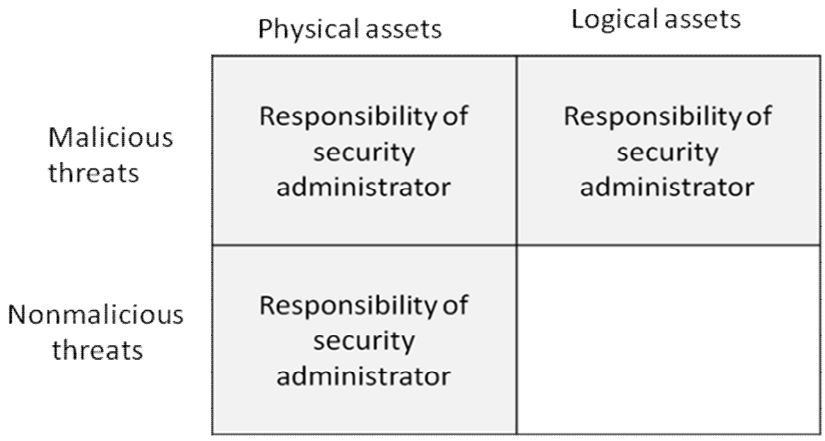


Figure 2-2 Is security administration responsibility

The information systems assets we must protect via security measures can be classified in two ways (Figure 2-1). The physical assets comprise personnel, hardware (including storage media and peripherals), facilities, supplies, and documentation. The logical assets comprise data/information and software. In a sense, this entire book deals with measures auditors can use to ensure these assets are secure.

In this lesson, however, we focus on the work usually performed by information systems security administrators. Although their specific functions vary across organizations, they tend to be responsible for controls over (1) both malicious and non-malicious threats to physical assets and (2) malicious threats to logical assets (Figure 2-2). In addition, they often are responsible for controls of last resort. These controls comprise backup and recovery procedures and insurance. They are invoked when all else fails.

We begin our discussion of security administration by examining the major functions that security administrators perform during the conduct of a security program. Next we consider some major threats to security that security administrators must consider and some controls they might implement to reduce expected losses from these threats. We then examine controls of last resort. Finally, we consider where information systems security administrators should be placed within the organizational hierarchy if they are to be able to perform their role effectively and efficiently.

2.2 Conducting a security program

A security program is a series of ongoing, regular, periodic reviews conducted to ensure that assets associated with the information systems function are safeguarded adequately. The first security review conducted is often a major exercise. Security administrators have to consider an extensive list of possible threats to the assets associated with the information systems function, prepare an inventory of assets, evaluate the adequacy of controls over assets, and perhaps modify existing controls or implement new controls. One outcome of the initial security review might be

a security policy to guide security practices within an organization and to provide a basis for subsequent evaluation of these practices., Subsequent security reviews might focus only on changes that have occurred for example, acquisition of a new mainframe computer, distribution of microcomputers to more sites, construction of new facilities, establishment of a new local area or wide area network, or the emergence of a new security threat. A security policy, if one exists, might need to be updated in light of these subsequent reviews.

Several formalized approaches have been proposed to undertake security reviews. For example, the United Kingdom Government's Central Computer and Telecommunications Agency has developed an approach called CRAMM (CCTA's Risk Analysis and Management Methodology), which is used as a government wide, standard approach to risk analysis and security management. Software also has been designed and implemented to support each stage of the approach. Baskerville provides an overview of the different security-evaluation approaches and their associated decision support software.

FIGURE 2-3 Major steps in the conduct of a security program.

Auditors must evaluate whether security administrators are conducting on-going, high-quality security reviews. If the security of information systems as-sets is at risk, asset safeguarding and data integrity objectives can be under-mined. For example, fire could destroy a mainframe facility, or employees might damage storage media to cover up evidence associated with frauds they have perpetrated. Similarly, system effectiveness and efficiency objectives can be undermined. For example, the destruction of an application system's documentation might mean programmers have difficulty modifying the system to accommodate changing user needs. Likewise, the destruction of hardware could mean required system response times cannot be achieved until the hardware is replaced.

In this light, the following sections describe eight major steps to be under-taken when conducting a security review (Figure 2-3): (1) preparation of a project plan, (2) identification of assets, (3) valuation of assets, (4) threats

identification, (5) threats likelihood assessment, (6) exposures analysis, (7) controls adjustment, and (8) report preparation. As we study these sections, our objective is to gain sufficient understanding of the steps in a security review for us to be able to evaluate how well they are performed by security administrators.

2.2.1 Preparation of a Project Plan

It might seem an obvious requirement that security reviews should commence with the preparation of a project plan. Unfortunately, security reviews are a mine field for the unwary. Security administrators, and any project team established to assist them, can get bogged down in detail unless strict constraints are imposed on the conduct of the review. If the review's objectives are not kept clearly in mind, too much work will be undertaken that has only marginal benefits. In due course this detail might be appropriate, but at the outset security administrators might wish to adopt a phased approach to the conduct of the overall security review program. Initial reviews focus on critical areas; subsequent reviews then address lesser concerns.

The project plan for a security review should encompass the following items:

Project Plan Component	*Explanation*
Objectives of the review	The objectives of the security review can be broadly based or narrowly defined. For example, the review might be undertaken to improve physical security over mainframe hardware in a particular division or to examine the adequacy of controls in light of a new threat to logical security that has emerged.
Scope of the review	The objectives of the review define, in part, the scope of the review. Defining scope is especially important, however, if the information systems function is widely dispersed throughout an organization. For example, if the adequacy of controls over hardware is to be evaluated, the scope of the review might or might not include all sites where microcomputers are located.
Tasks to be accomplished	Although the overall tasks to be undertaken will be known, specific tasks must be defined. For example, the project plan might specify the tasks to be undertaken to compile an inventory of hardware at end-user sites.

Project Plan Component	*Explanation*
Organization of the project team	Depending on the size and complexity of the review, the security administrator might have to enlist the assistance of consultants or staff who have detailed knowledge of the areas to be evaluated.
Resources budget	The resources budget will be affected by the size and complexity of the review. It must specify the labor hours, materials, and money required to complete the review.
Schedule for task completion	The plan must specify which tasks must be completed by what dates if the objectives of the security review are to be accomplished on a timely basis.

The security-review plan should be documented formally to provide working guidelines for the project team. Standard tools such as Gantt charts and PERT charts can be used to assist the documentation and communication processes. As the project progresses, security administrators must monitor progress of the review against the plan.

2.2.2 Identification of Assets

The second major step in a security review is to identify the assets associated with the information systems function. This step can be difficult for two reasons. First, some organizations possess a substantial number of information systems assets. For example, a large organization might have several mainframe computers, hundreds of microcomputers, a wide-area communications network, many local area communication networks, and thousands of files and programs. Second, the information systems assets might be widely distributed throughout an organization. For example, they could be scattered across different divisions and departments, and many might be located with end users.

One way to identify assets is to seek out instances within various general categories. For example, security administrators might use\the categories shown in Figure 2-1:

Asset Category	*Examples*
Personnel	End users, analysts, programmers, operators, clerks, guards.
Hardware	Mainframe computers, minicomputers, microcomputers, disks, printers, communications lines, concentrators, terminals.
Facilities	Furniture, office space, computer rooms, tape storage racks.
Documentation	Systems and program documentation, database documentation, standards, plans, insurance policies, contracts.
Supplies	Negotiable instruments, preprinted forms, paper, tapes, cassettes.
Data/information	Master files, transactions files, archival files.
Applications software	Debtors, creditors, payroll, bill-of-materials, sales, inventory.
Systems software	Compilers, utilities, database management systems, operating systems, communications software, spreadsheets.

Within each category, the review team must prepare a comprehensive list of assets. The difficult part knows the level of aggregation at which to work. For example, consider the problem of preparing an inventory of application programs in a large organization that has several thousand programs. On the one hand, for backup purposes a complete inventory of these programs will be needed. On the other hand, from the viewpoint of deciding what controls should be exercised over the disks on which they are stored, it might be possible to consider groups of programs rather than individual programs.

Similar problems exist with data files. Do data assets need to be identified at the data item, group, record, or file level? Clearly, the finer the level of asset identification required, the more costly will be the review process.

2.2.3 Valuation of Assets

The third step in a security review, valuing the assets, is also a difficult step. Parker points out that the valuation might differ depending on who is asked to give the valuation, the way in which the asset could be lost, the period of time for which it is lost, and the age of the asset (Figure 2-4). In terms of who values the asset, an asset might be more useful to some people than to others. For example, end users who employ a generalized retrieval package more frequently than programmers are more likely to assign a higher value to the package. In terms of how the asset is lost, accidental loss might be less serious than loss that arises through an irregularity. For example, although the accidental destruction of a customer master file might be serious, management might be more concerned if it is stolen by a competitor. In terms of the time period of loss, for most assets the loss becomes more serious as use of the asset is denied for a longer period. For example, if it is difficult to replace a piece of hardware quickly, management might value it more highly than other hardware that has a higher capital cost

but that, nonetheless, can be replaced immediately. In terms of the age of the asset, most assets deteriorate with age. For example, management might be less concerned about a competitor gaining access to an old customer master file.

Valuation of physical assets also cannot be considered in isolation from valuation of logical assets. The reason is that the value of physical assets often must be considered in light of the logical assets they store. For example, consider the value of a microcomputer's hard disk. The replacement value of the physical disk might be only a few hundred dollars. Its contents, however, might be worth thousands, even millions of dollars. The project team must take care, therefore, to prepare an inventory of all material physical and logical assets that fall within the scope of the review. Otherwise, the team might fail to identify significant exposures.

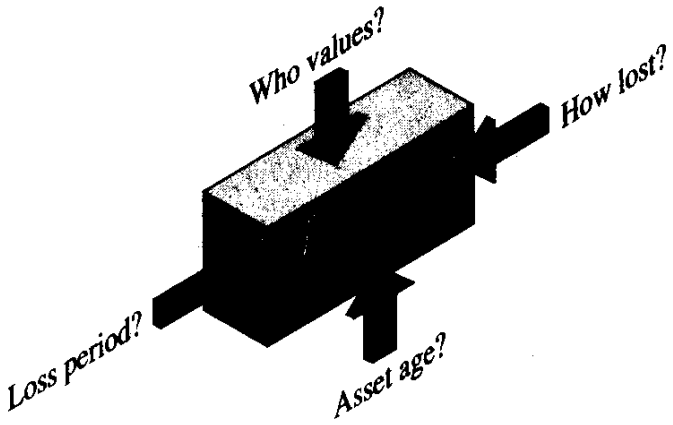


FIGURE 2-4 Factors that affect the valuation of information systems assets.

Several techniques can be used to assign a value to an asset. In some cases, users might be able to provide a direct dollar valuation for the asset. For example, if an item of hardware can be replaced quickly, they simply might value it at its acquisition cost, assuming the item does not store logical assets whose value must also be taken into account. Often, however, precise dollar valuations are hard to assign to assets. For example, it can be difficult to determine the loss of customer goodwill that occurs when system failure leads to degradation in service or to estimate the lost revenues that result when a competitor steals proprietary software. In these types of situations, the following sorts of indirect valuation techniques might be used:

1. A formal procedure, such as the Delphi method, can be used to try to get stakeholder consensus on an asset valuation. The Delphi method employs successive rounds of confidential questionnaires to elicit the respondents' opinions on some matter. Feedback is provided after the results of each round are obtained in case respondents then wish to revise their views.

2. Users can be asked to value an asset on some type of scale where, say, a score of 1 represents a low value and a score of 10 represents a high value. This approach might help users rank assets in order of importance.
3. Courtney suggests that users be asked to value assets on a logarithmic scale. They assign a rating, v, to an asset based on their estimate of the dollar value as a function of the base 10. Thus, an asset valued around $100 would be assigned a v of 2; an asset valued around $1 million would be assigned a v of 6. Using this technique, respondents are not forced into making fine discriminations between assets based on value.

When undertaking the asset-valuation task, security administrators must be careful that the evaluation does not flounder because it becomes too onerous. The primary objectives of asset valuation are to develop users' sensitivity to the possible consequences of a threat that eventuates and ultimately to enable an estimate to be made of the amount that can be justified as expenditure on safeguards. The task of asset valuation should be pursued only to the extent that these objectives can be accomplished satisfactorily.

2.2.4Threats Identification

A threat is some action or event that can lead to a loss. During the threats-identification phase, security administrators attempt to flesh out all material threats that can eventuate and result in information systems assets being exposed, removed either temporarily or permanently, lost, damaged, destroyed, or used for unauthorized purposes.

One useful way to identify threats is first to consider possible sources of threats and then to consider the types of threats these sources might initiate (Figure 2-5). When considering the types of threats, it is also useful to categorize threats as malicious (intent to do harm) or nor-malicious (accidental). For example, the following threats arise from sources that are external to the organization:

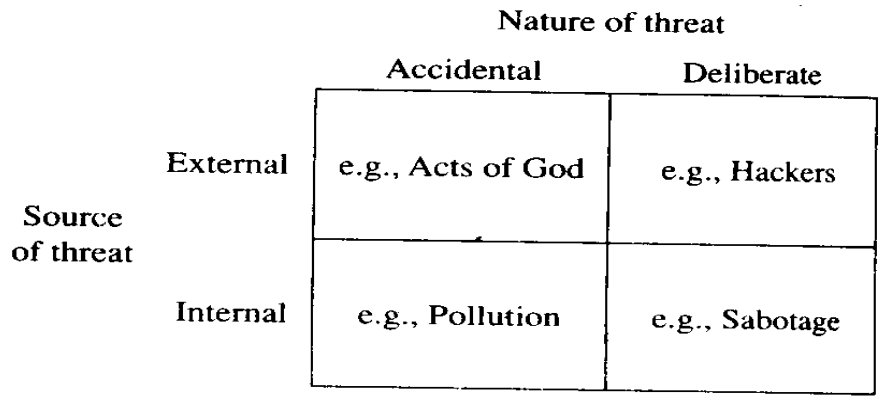


FIGURE 2-5 Types of threats facing information systems assets.

Source of Threat	*Examples of Threat Types*
Nature/Acts of God	Earthquake, flood, fire, mud, gases, projectiles, living organisms, extreme temperatures, electromagnetic radiation.
Hardware suppliers	Unreliable hardware, ineffective hardware, incompatible hardware, improper maintenance, lawsuits.
Software suppliers	Erroneous software, ineffective software, poor documentation, improper maintenance, lawsuits.
Contractors	Erroneous software, ineffective software, improper hardware/software maintenance, untimely provision of services, disclosure of confidential information.
Other resource suppliers	Power outages, disruption to communication services, untimely provision of resources.
Competitors	Sabotage, espionage, lawsuits, financial distress through fair or unfair competition.
Debt and equ'ty holders	Financial distress though foreclosure on claims.
Unions	Strikes, sabotage, harassment.
Governments	Financial distress through onerous regulations.
Environmentalists	Harassment, unfavorable publicity.
Criminals/hackers	Theft, sabotage, espionage, extortion.

Similarly, the following threats arise from sources that are internal to the organization:

Source of Threats	*Examples of Threat Types*
Management	Failure to provide resources, inadequate planning and control.
Employees	Errors, theft, fraud, sabotage, extortion, improper use of services.
Unreliable systems	Hardware failure, software failure, facilities failure.

Some types of threats are clear because their consequences are immediate and obvious. For example, the nature of and impact of fire on information systems assets usually will be readily apparent. Other types of threats are more subtle. For example, astute actions by a competitor can undermine the financial viability of an organization. As a result, controls might not be maintained because management initiates cost-cutting measures.

2.2.5 Threats Likelihood Assessment

Having identified the threats that face the information systems function, security administrators must next attempt to estimate their likelihood of occurrence of each threat over a specified time period. In some cases, statistical data might be available. For example, an insurance company

might be able to provide information on the probability of a fire occurring over varying time periods. Similarly, analyses have been undertaken on cases of computer abuse that give insights into their likelihood of occurrence.

Often, however, prior data is not available. Security administrators must then elicit the likelihood of occurrence of a threat from the stake holders associated with an information system. For example, users can probably best estimate the likelihood of erroneous data leading to decisions that incur significant losses, and management can probably best estimate the likelihood of controls being compromised because financial distress has arisen. As with asset valuation, security administrators can use formal elicitation techniques like the Delphi method to obtain estimates of the likelihood of occurrence of a threat.

To some extent, the nature and value of the assets associated with the information systems function affect the likelihood of occurrence of a threat. If the information systems function has many high-value, proprietary software packages, for example, it is a prime target for piracy attempts. Thus, the identification and valuation of assets also assists with the identification of threats and their likelihood of occurrence.

Periodically, we must reassess the likelihood of a threat occurring. Changes can occur in the structure, direction, and environment of an organization that produce changes in the threat profile that face an organization.

2.2.6 Exposures Analysis

The exposures analysis phase comprises four tasks: (1) identification of the controls in place; (2) assessment of the reliability of the controls in place; (3) evaluation of the likelihood that a threat incident will be successful, given the set of controls in place and their reliability; and (4) assessment of the loss that will result if a threat incident circumvents the controls in place (Figure 2-6). When these tasks are accomplished, the exposures associated with the information systems function can be determined. An exposure is simply the expected loss that will occur over some time period, given the reliability of the controls in place. Exposures arise because either (1) there is no control to cover the threat incident or (2) there is some probability that the control in place will not operate reliably for the particular threat incident that occurs (Figure 2-7).

Consider, then, the first task: identifying the controls in place. Perhaps the easiest way to perform this task is to use one of the many questionnaires designed to assess security. These questionnaires contain extensive control checklists that security administrators can use to determine systematically whether a control is missing. Like auditors, security administrators can use interviews, observations, and documentation to obtain information about the controls in place.

Identify the control in place
Assess the reliability of the control in place
Evaluate the likelihood a thread will be successful
Assess the resulting loss if the threat is successful

FIGURE 2-6. Major tasks in the exposures analysis phase.

To assess the reliability of these controls, security administrators must test them. In some cases, tests are straightforward. For example, usually it is easy to determine whether locked doors prevent unauthorized access to a computer room. Some types of controls, however, are difficult to test. For example, to check whether fire extinguishers work, usually one would not start a fire and then discharge the extinguisher to evaluate its effectiveness. Instead, auditors must rely on assurances given by the manufacturer and examine the extinguisher's servicing tag to see whether it has been maintained regularly. Similarly, testing whether the fire extinguisher system for a computer room works can cost many thousands of dollars. In some high-exposure situations auditors might periodically simulate a fire in the computer room to determine whether the system extinguishes the fire. Usually, however, auditors rely on maintenance records and the manufacturer's assurances that the system will operate effectively.

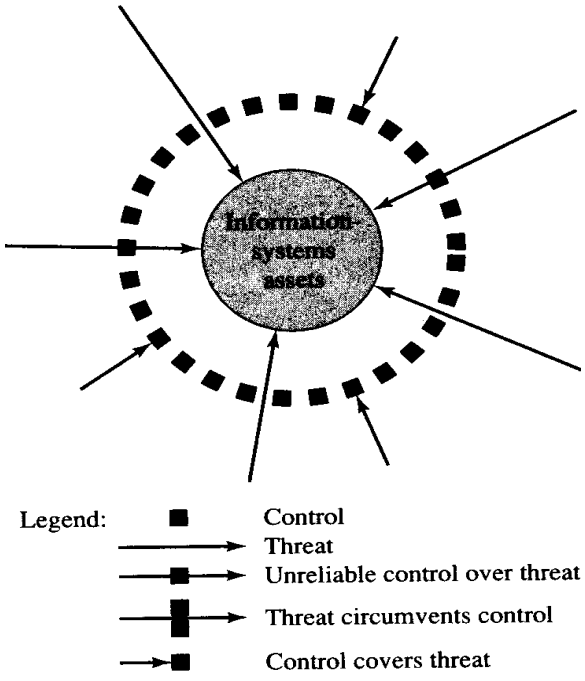


FIGURE 2-7 Threats, control reliability, control coverage, and exposures.

To evaluate the likelihood of a threat incident circumventing a control, the security administrator considers each of the assets or categories of assets identified during the second phase of the review, considers each of the threats identified during the fourth phase of the review, determines whether a control exists to cover the threat incident, and, if so, evaluates the probability of the control operating effectively to eliminate or mitigate the effects of the threat incident. To aid in this task, Parker recommends that security administrators write scenarios to describe how threat incidents could compromise controls. These scenarios then can be considered by stakeholders to assess their realism (Table 2-1).

To determine the losses that will be incurred from a threat incident that circumvents controls, the effect of the threat incident should first be determined. Will the asset be lost, damaged, exposed, removed, destroyed, or used for unauthorized purposes? Next, a value must be assigned to the effect. The value of the asset determined in the third phase of the review provides the basis for this assessment. Security administrators must determine whether the full value of the asset will be lost if the threat is successful or whether the loss will be partial.

Table 2-1 Scenario analysis of exposures

Threat:	Malicious damage to mainframe computer by operator.
Existing controls:	1. Two operators always present. 2. Background check carried out before hiring operators.
Control weaknesses:	Ongoing psychological stability of operators is not assessed regularly in a formal way.
Scenario:	An operator who becomes psychologically unstable could destroy the mainframe when the other operator leaves the computer room temporarily. Alternatively, the unstable operator might be able to overpower the other operator to carry out the damage.
Probability of occurrence:	Less than 1 in 10,000 a year.
Loss that would occur:	$600,000 replacement cost of machine. Backup exists for software and data.
Assessment:	Exposure is acceptable as costs of carrying out psychological testing exceed $60 per year. The effect of this type of testing on employee morale is also a serious concern.

For each asset and each threat, the expected loss can then be calculated using the following formula:

$$EL = p_t \times p_f \times L \text{ where:}$$

EL = expected loss associated with asset (exposure)

p_t = probability of threat incident occurring

p_f = probability of control failure

L = resulting loss if threat is successful

For example, given the controls that exist, if the likelihood, p_t, that a fire will occur in a corporation's computer room in any one year is .001, the probability, p_f, of controls failing to detect and extinguish the fire is .1, and the loss, L, that will occur as a result is $4 million, the expected loss each year, EL, with respect to the fire threat and the computer room assets is $400; that is, EL = .001 X .1 X $4,000,000. The exposure, therefore, is $400.

If security administrators have not chosen the right level of aggregation in their identification of assets and threats, the exposures-analysis phase may falter badly. Clearly, substantial work can be involved in carrying out an exposures analysis. Too much detail will undermine the analysis. Security administrators need to devote most effort to material assets and material threats. In this regard, the choices they make during prior phases of a security review about the assets and threats that are important have significant implications for the conduct of the exposures-analysis phase.

2.2.7 Controls Adjustment

Subsequent to the exposures analysis, security administrators must evaluate whether the level of each exposure is acceptable. Formally, this evaluation means they must determine whether over some time period any control can be designed, implemented, and operated such that the cost of the control is less than the reduction in expected losses that occurs by virtue of having the control in place and working to cover one or more threats. In other words, the benefits of a control that arise because it reduces expected losses from threats must exceed the costs of designing, implementing, and operating the control.

How security administrators make this decision is in large part a matter of judgment, experience, and training. To some extent, guidance can be obtained from the control questionnaires used to identify missing controls during the exposures-analysis phase. These controls are candidates for inclusion in a revised controls system. Security administrators also might consult their colleagues in other organizations to determine control profiles that are used commonly. These profiles represent the combined experience and judgments of others who have faced similar problems. As such, they could provide important insights into controls that will be cost-effective.

2.2.8 Report Preparation

The final phase in a security review is the preparation of a management report. This report documents the findings of the review and, in particular, makes recommendations as to new safeguards that should be implemented and existing safeguards that should be terminated or modified.

Like all reports to management, often the most difficult part is getting the recommendations accepted. The level of acceptance depends on the extent to which management agrees with the criticality of the exposures identified and whether they perceive the recommended safeguards are economically, technically, and operationally feasible. Again, scenarios are a useful technique for increasing management's sensitivity to the exposures identified. They provide a tangible basis for management to evaluate how concerned they should be about an exposure. With respect to the feasibility of the safeguards recommended, the onus is on security administrators to demonstrate that the safeguards are within the information systems function's capabilities to design, implement, and operate.

The security report also must include a plan for implementing the safeguards recommended. Both the seriousness of the exposure to be rectified and the difficulty of implementing the remedial safeguards must be considered. The most serious exposures should be addressed first, but then the ease with which a safeguard can be installed or modified should determine the order of implementation. To the extent some safeguards are interdependent and management decides not to implement them all, the report must consider alternative control configurations.

2.3 Major security threats and remedial measures

The previous section describes a general methodology for evaluating security over information systems assets and selecting and implementing controls. This section briefly discusses some of the major security threats that face the information systems function (Figure 2-8) and safeguards that can be implemented. Auditors must understand the nature and potential consequences of these threats for audit objectives and the controls that are likely to be effective. If an auditor deems controls over these threats to be inadequate or unreliable, the extent of substantive testing will have to be expanded, especially in relation to asset safeguarding and data integrity objectives.

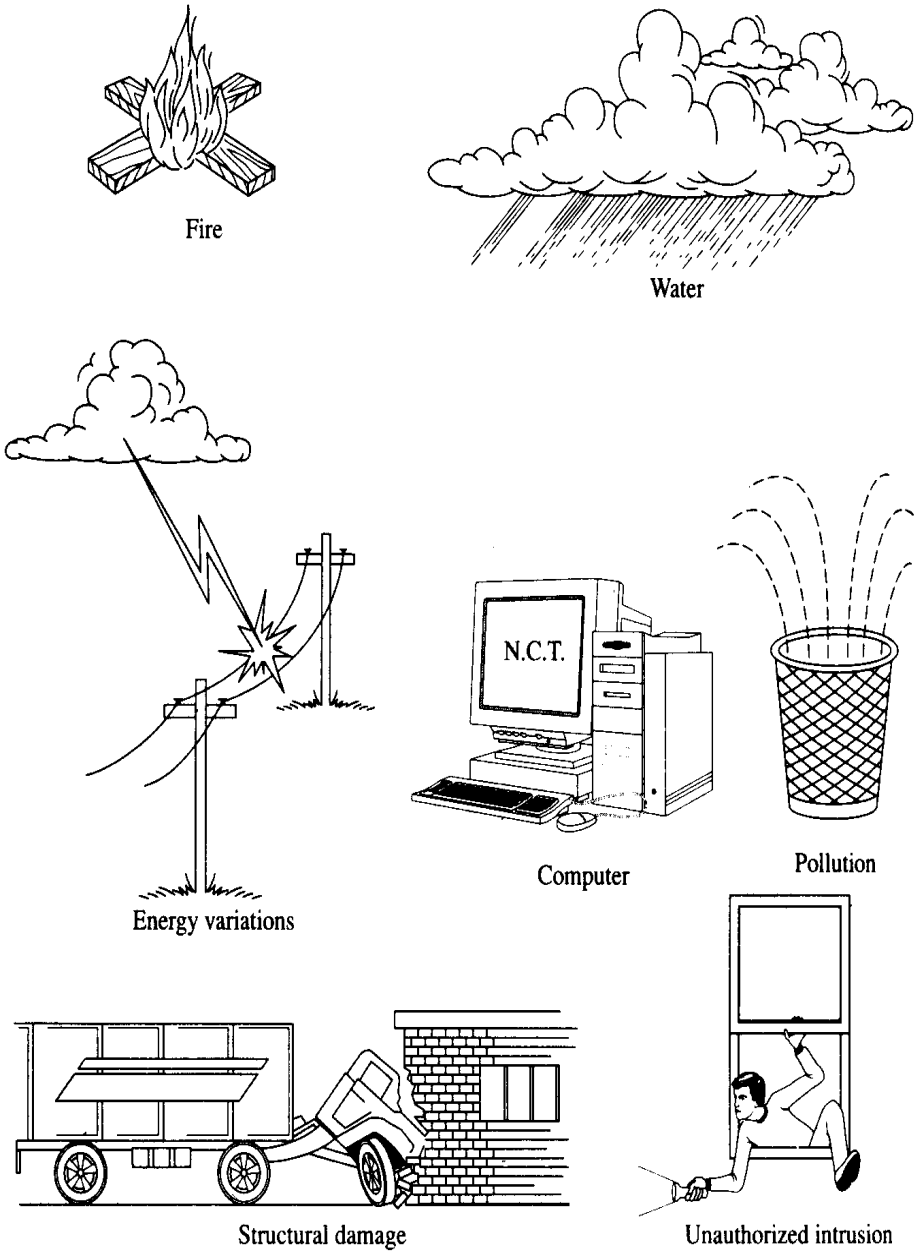


FIGURE 2-8. Major threats to the information systems function.

2.3.1 Fire Damage

Some countries have various public service and governmental organizations that provide advice on fire-protection measures. Nevertheless, implementing a specific fire-protection system usually requires specialist advice. Some major features of a well-designed fire-protection system follow:

1. Both automatic and manual fire alarms are placed at strategic locations throughout those parts of the organization where material information systems assets are sited.

2. Automatic fire-extinguisher systems exist at strategic locations that dispense the appropriate suppressant: water, carbon dioxide, or halon. Water is the least harmful suppressant for humans; however, it can damage equipment. Carbon dioxide is relatively cheap, but it can debilitate a person in seconds. Halon gas is a widely used suppressant because it is safe and effective. Personnel will not be debilitated by halon, for example, providing only short-term exposure occurs. Moreover, it is noncorrosive, nonconductive, highly compressible, and chemically stable. In some countries, however, production of halon gas is now prohibited because of its detrimental effect on the earth's ozone layer. Alternative suppressants are being sought.

3. Appropriate types of manual fire extinguishers exist at strategic locations throughout those parts of the organization where material information systems assets are sited.

4. A control panel shows where in the organization an automatic or a manual alarm has been triggered.

5. Beside the control panel, master switches exist for power (including air conditioning) and automatic extinguisher systems.

6. Buildings where material information systems assets are sited have been constructed from fire-resistant materials, and they are structurally stable when fire damage occurs.

7. Fire extinguishers and fire exits are marked clearly and can be accessed easily by staff.

8. When a fire alarm is activated, a signal is sent automatically to a control station that is always staffed.

9. Good housekeeping procedures ensure that combustible pollutants and materials are minimized around high-value information systems assets for example, computer rooms are cleaned regularly to remove paper lint, and printer paper supplies are kept in a separate room.

10. To reduce the risk of extensive damage from electrical fires, electrical wiring should be placed in fire-resistant panels and conduit.

Security administrators should arrange regular inspections and tests of all fire protection systems and ensure that they are properly serviced. Proper use of these systems also requires staff training and periodic drills. The procedures to be followed during an emergency also should be documented.

2.3.2 Water Damage

Water damage to information systems assets can be the outcome of a fire; for example, an extinguisher system sprays water that enters hardware, or water pipes could burst. Water damage can result, however, from other

sources: cyclones, tornadoes, ice, and torrential rains. In 1974, for example, the city of Brisbane experienced freak flooding after torrential rains. As a result, the Brisbane River burst its banks. One mainframe computer room in close proximity to the river was submerged completely.

Some major ways of protecting information systems assets against water damage follow:

1. Where possible, have waterproof ceilings, walls, and floors.
2. Ensure that an adequate drainage system exists.
3. Install alarms at strategic points where material information systems assets are located.
4. In flood areas, have all material information systems assets located above the high-water level.
5. Have a master switch for all water mains.
6. Use a dry-pipe automatic sprinkler system that is charged by an alarm and activated by the fire.
7. Cover hardware with a protective fabric when it is not in use.
8. To prevent flooding, locate information systems assets above the ground floor of the buildings in which they are housed
9. Have a dry moat around buildings where material information systems assets are housed.

Again, regular inspections, tests, and drills are essential if the disaster plan is to be operational when a situation of potential water damage arises.

2.3.3 Energy Variations

Energy variations take the form of increases in power (surges or spikes), decreases in power (sags or brownouts), or loss of power (blackouts). They can disrupt not only hardware operations but also the systems needed to maintain an acceptable operational environment one that is dust free and relatively constant with respect to temperature and humidity. Thus, careful assessment of the likelihood of unacceptable energy variations occurring is essential to the ongoing operations of the information systems function. Moreover, energy sources must be monitored constantly to ensure their continuing adequacy and reliability.

To protect hardware against temporary increases in power, voltage regulators can be used. To protect hardware against sustained increases in power, circuit breakers can be used. A wide range of voltage) regulators and circuit breakers can now be purchased for mainframe computers, minicomputers, and microcomputers.

To protect hardware against power loss, if possible, two independent supply sources should exist so that one can be used if the other fails. In addition, uninterruptible power supply systems (UPSs) can be installed. Three types are available. Static UPSs rely primarily on batteries and are intended as

short-term backup. Rotary UPSs provide a generator as backup. Usually they are driven by diesel engines and are intended for longer-term power loss. Hybrid UPSs employ both batteries to provide power initially and a generator that takes over when the batteries run low. Like rotary UPSs, they are intended for longer-term power loss. Unless sustained operation of a microcomputer is critical to an organization's operations, usually UPSs are purchased to support only mainframe computers and minicomputers.

The design of security for the information systems function must provide for the possibility of total loss of power. For example, certain controls such as doors can fail-safe on a power loss. It must be possible to deactivate these controls manually should staff need to exit the building quickly. Other safeguards such as alarms and extinguisher systems also might fail to operate in the event of power loss. Alternative controls must then be used. For example, buildings might be evacuated and locked to protect personnel and to reduce the likelihood of unauthorized access to and use of hardware, software, and data.

2.3.4 Structural Damage

Structural damage to information systems assets can occur in several ways: earthquake, wind, mud, snow, avalanche, and mishap. Structural damage can also be an outcome of some other disaster. For example, fire might weaken the floor of a room in which mainframe computers and their associated peripherals are housed. Eventually the floor might collapse, and hardware might be damaged as a result.

Some information systems assets are more prone to structural damage than others. Those located in an earthquake region, for example, face a higher risk. Similarly, smaller assets, like microcomputers, are prone to mishap. A microcomputer might be dropped, for example, as it is carried from one desk to another. The microcomputer's hard disk drive, as well as other components, could be damaged.

Preventing disaster from occurring through structural damage is often an engineering problem. In the design of a building, for example, the engineers will consider the structural stresses the building might endure. They should be notified, however, if significant computer hardware is to be housed in the building. They can then take this fact into consideration when they prepare their building plans and strengthen, say, the floor areas where equipment will be located.

If there is some choice as to where information systems assets are to be located, the site chosen should be the least prone to structural damage for example, away from a floodplain or an earthquake region. Similarly, information systems assets should be located, if possible, on an upper floor of a building. They are less susceptible to damage by floods.

Information systems assets also should be secured so they will not dislodge or tip easily. For example, microcomputers should be placed on stable desks that are not located near major thoroughfares. Similarly, mainframe

hardware and storage cabinets should be secured so they will not overturn if structural stress is placed on a building during, say, an earthquake.

2.3.5 Pollution

The ongoing operations of much of the equipment used by the information systems function depend on having an environment that is relatively unpolluted. Pollution can damage a disk drive, for example, and as a result critical data could be lost. Valuable time also could be lost while the damaged equipment is repaired. Pollution can also cause fires. Toigo reports that one major information systems insurer attributes a significant percentage of fires to pollution-related spontaneous internal combustion associated with hardware components.

The major pollutant is dust. Dust can become a problem if the air passing through the air conditioning system is filtered inadequately or if it is allowed to accumulate on, say, floors and ceilings. More subtle forms of pollution exist, however. For example, coffee is a pollutant if it is spilled in a microcomputer keyboard or printer, thereby rendering the keyboard or printer inoperable.

Several steps can be taken to reduce losses from pollution. Ceilings, walls, floors, storage cabinets, and equipment should be cleaned regularly. Vacuuming is especially important, particularly in areas where dust collects, such as under raised floors. Dust collecting rugs can be placed at entrances. Floors can be treated with special antistatic compounds. Dust generating activities for example, paper shredding, decollation, or bursting should be carried out well away from dust-sensitive equipment. Foodstuffs can be prohibited in certain areas such as mainframe computer rooms or microcomputer work areas, especially where pests such as rodents and insects can cause problems. Regular emptying of wastepaper baskets also prevents dust and combustible materials from collecting.

2.3.6 Unauthorized Intrusion

Unauthorized intrusion takes two forms. The intruder physically could enter an organization to steal information systems assets or carry out sabotage or extortion. For example, intruders might be seeking to remove magnetic tapes, disk and tape cartridges, or diskettes or to plant a bomb. Alternatively, the intruder might eavesdrop by wiretapping, installing an electronic bug, or using a receiver that picks up electromagnetic signals. One other form of eavesdropping is visual eavesdropping. The intruder might photograph sensitive information or use a telescope to view the information.

Physical intrusion can be inhibited or prevented by erecting various barriers. Buildings that house information systems assets can be protected by a wall, a fence, or a dry moat. Doors and windows should be secured. Some type of card locking system might then be used, for example, to restrict entry to authorized personnel. Sometimes air conditioning ducts allow unauthorized entry to buildings. Intruders simply have to gain access to the roof of the building (perhaps via the fire escape) and crawl through the ducts. Thus,

security administrators must ensure that air conditioning ducts are secure. Buildings that house high-value information systems assets, such as an organization's mainframe computer installation, might be disguised to reduce the likelihood of unauthorized intrusion occurring.

Alarms and guards can be used to detect unauthorized intruders. Various types of security devices and systems can be installed that signal the presence of an intruder. Nevertheless, these defenses can be compromised: a guard could be bribed or a security device improperly deactivated. The last lines of protection are then safes, vaults, filing cabinets, or locks. For example, magnetic tapes, disk and tape cartridges, and diskettes can be locked in filing cabinets, microcomputer components can be secured using lockdown devices, and check stationery (negotiable instruments) can be stored in a safe. Nonetheless, even these controls might not withstand the threats posed by a saboteur or terrorist intent on destruction.

Intruders also could pose as visitors or employees to attempt unauthorized entry to buildings that house information systems assets. Receptionists or guards can challenge unidentified visitors and provide advance warning of unauthorized intrusion. A badge system can be used to identify the status of personnel within the building: permanent staff or visitors. All visitors should be escorted by a permanent staff member. Unescorted visitors or persons without a badge should be questioned about their presence in the building. A security check might be performed before visitors are issued a badge.

Eavesdropping breaches the privacy of data. Intruders seek access to sensitive information such as passwords, sales information, geological survey information, and engineering design information.

Controls must be implemented to prevent eavesdropping via the electromagnetic emanations that computer equipment produces. For example, eavesdropping devices can be used to pick up emissions from microcomputers or printers. Some manufacturers shield their computer equipment against electromagnetic emanations. Alternatively, equipment must be housed in buildings that are designed to inhibit emissions. For example, specially screened rooms can be built to prevent emanations from equipment that processes highly sensitive data.

Eavesdropping can also be undertaken via bugs that are installed by intruders. Various devices are available to detect the presence of bugs. Security administrators periodically might employ a security firm that possesses these devices to check whether bugs are present.

In a communications network the points most likely to be wiretapped are the junction boxes and the private branch exchange. It is difficult to wiretap a communications line after it leaves the building in which the computer is housed; the line might be underground, signals might be sent via microwave, several thousand channels might be multiplexed together. Thus, security administrators should ensure that the junction boxes and private branch exchange are secure.

Visual eavesdropping can be prevented in several ways. Cameras should not be allowed in buildings where security is a concern. Some buildings have no windows; however, staff might object to the absence of natural light. If windows do exist, they can be shielded by blinds or curtains. Visual display units can also be placed strategically so intruders outside a building cannot use telescopes or cameras with a telescopic lens to view or photograph output.

2.3.7 Viruses and Worms

A virus is a program that requests the operating system of a computer to append it to other programs (Figure 2-9). In this way the virus propagates to other programs. Viruses can be easily transmitted, for example, via files that contain macros that are sent as attachments to electronic-mail messages. For example, macros in word processing files can be infected with viruses. When recipients open these infected files with their own copies of the word processing software, their system will be infected unless their word processing software first checks for the presence of these viruses.

A virus can be relatively benign; for example, it can cause minor disruptions by printing humorous messages or drawing diagrams on visual-display screens. Alternatively, it might be malignant; for-example, it could delete files, corrupt other programs so they are unusable, or severely disrupt the operations of application systems.

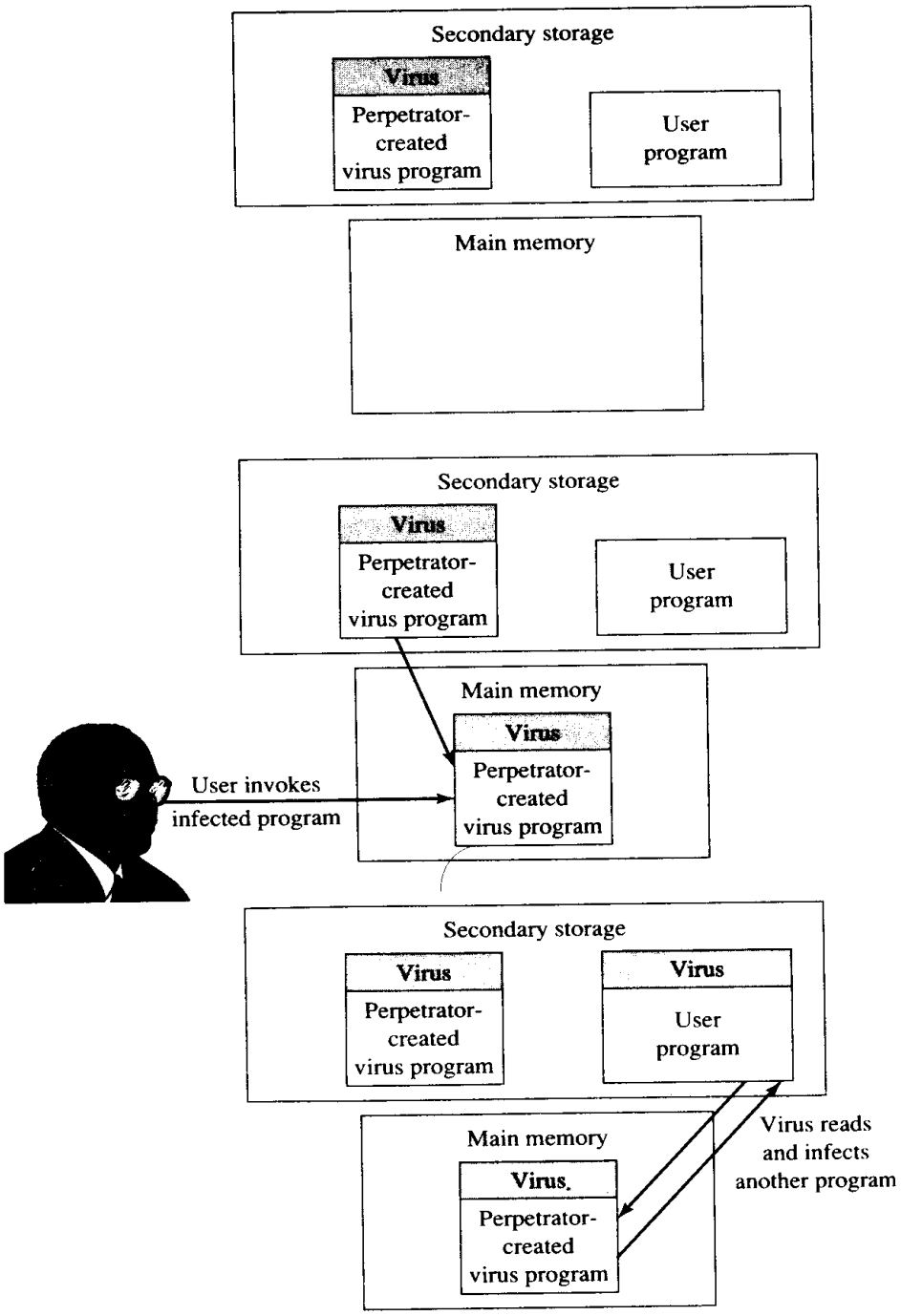


Figure 2-9 Program corruption through viruses

To reduce expected losses from viruses, security administrators can implement the following types of controls:

Type of Control	*Examples*
Preventive	Use only "clean," certified copies of software files or files that contain macros.
	Do not use public domain/shareware software or files that might contain macros that are prepared by others unless they have been checked for viruses.
	Download software or files that contain macros only from reputable bulletin boards or Web sites or avoid downloading from bulletin boards or Web sites altogether.
	Implement read-only access over software.
	Check new software with antivirus software before it is installed.
	Check new files with antivirus software before they are used.
	Educate users about the dangers of viruses and the means of preventing infection.
Detective	Regularly run antivirus software to detect infections. Undertake file size comparisons to determine whether the size of programs has changed. Undertake date/time stamp comparisons to determine whether unauthorized modifications have been made to software.
Corrective	Ensure "clean" backup is maintained.
	Have a documented plan for recovery from virus infections.
	Run antivirus software to remove infections.

Like viruses, worms propagate copies of themselves with benign or malignant intent. Whereas viruses attach themselves to other legitimate programs, however, worms usually exist as separate, independent programs. Worms use operating system services as their means of replication. Often they exploit some type of bug or security weakness in the operating system to infiltrate other systems. For example, a worm introduced into the Internet computer network in 1988 exploited a bug and a security weakness in the UNIX operating system and e-mail facilities to propagate rapidly to machines connected to the network. The worm caused major disruption as machines in the network labored to execute the worm's many replicas. Machines had to be taken off line and purged of all copies of the worm.

Threats from worms arise when an organization connects its computers to an open network one that users can join relatively easily and one in which they are not subjected to rigorous security requirements. Use of open networks is likely to increase because many advantages accrue from easy access to other computers for example, facilitating use of e-mail, electronic commerce, and the World Wide Web.

Unfortunately, establishing effective controls over worms is often more difficult than establishing effective controls over viruses. Exposures that arise from viruses can be controlled to a large extent by actions the organization takes itself. Exposures that arise from worms, however, must be addressed

by all users of a network. Otherwise, control weaknesses in one user's system can undermine security in another user's system.

If security administrators work for organizations that participate in open • networks, they should undertake the following types of control steps:

1. Actively lobby to establish network security administration groups that will take responsibility for security in the networks in which their organization participates.
2. Participate in efforts to establish control protocols for network users to follow to reduce exposures from threats like worms.
3. Be aware of weaknesses in network resources that can undermine security. Implement any remedial measures that can cost-effectively reduce exposures from these weaknesses. For example, change any system default passwords and remove "visitor" accounts.
4. Ensure that strong access controls exist over those resources that are the security administrator's responsibility. For example, inform users of the importance of choosing passwords that are difficult to guess.
5. Educate users so they are aware of the importance of controls, the need for backup, and the actions they should take when they suspect a security violation has occurred.

2.3.8 Misuse of Software, Data, and Services

Organizations can incur losses because software, data, and services they own are misused. For example, the following types of abuses can arise:

1. Generalized software and proprietary databases that the organization develops are stolen by employees or competitors. The organization loses the revenue it would otherwise obtain from sales of the software and databases.
2. The organization fails to protect the personal privacy of individuals about whom it stores data in its databases. It may incur losses because of legislative breaches or unfavorable publicity.
3. Employees use information systems services to support their own personal activities. For example, they may use computer time for private consulting purposes or have the organization acquire resources (e.g., microcomputer hardware and software) for their own private use.

Security administrators can implement various controls to reduce expected losses from these types of abuses. In some countries (e.g., Australia, the United States, the United Kingdom), software is protected via copyright laws. Also, software sometimes can be protected via patents, licensing agreements, or trade-secret laws. Moreover, when the name of a software package is deemed important to its marketing, protection might be available through a trademark. Security administrators must evaluate each piece of software to determine whether it needs .to be protected, the form of protection needed, and the best means of providing protection.

Control can also be enhanced if security administrators take responsibility for or are informed about acquisition of hardware/software and enforcement of licensing agreements. Centralized control over hardware, software, data, and services reduces organizational flexibility, but it might be necessary to reduce expected losses from abuse to an acceptable level.

TABLE 2-2 Example Code of Conduct for Information Systems Personnel

---

1. Be honest, fair, and trustworthy.
2. Honor property rights with respect to information systems assets and take steps to preserve these property rights.
3. Respect a person's, groups, or organization's right to privacy and take steps to preserve privacy.
4. Respect confidentiality requirements and take steps to preserve confidentiality.
5. Respect the work of others and give proper credit when using the work of others.
6. Maintain professional competence.
7. Exercise due care when developing, implementing, and operating information systems.
8. Carefully evaluate the potential impact of systems.
9. Act ethically where deleterious impacts might arise from information systems work.
10. Seek and provide appropriate peer review of information systems work.
11. Respect access privileges assigned to information systems assets.
12. Keep management fully informed of material issues relating to information systems assets or the conduct of information systems work.

2.3.9 Hacking

A computer hacker is a person who attempts to gain unauthorized entry to a computer system by circumventing the system's access controls. Hackers can have benign or malignant intent. They simply might explore the capabilities of the system they hack or read files without changing them (computer trespass). Alternatively, they could wreak havoc by deleting critical files, disrupting system operations, or stealing sensitive data and programs.

In some countries, laws have been changed specifically to cover the activities of hackers (e.g., Australia, the United States). Because laws might not provide adequate remedial damages, however, it is better to prevent hacking in the first place. In this light, the primary preventive safeguards that security administrators can employ are access controls, especially logical access controls such as hard-to-determine passwords.

2.4 Controls of last resort

In spite of safeguards that might be implemented, the information systems function still could suffer a disaster. A control might fail, or a threat might occur that management has not considered or that management has decided to accept as an exposure that cannot be covered via cost-effective controls. When disaster strikes, it still must be possible to recover operations and mitigate losses. In this situation, two controls of last resort must take effect: (1) a disaster recovery plan and (2) insurance.

2.4.1 Disaster Recovery Plan

The purpose of a disaster recovery plan or contingency plan is to enable the information systems function to restore operations in the event of some type of disaster. The impact of a disaster might be localized; for example, a personal-computer user might accidentally delete critical data stored on a hard disk. The impact, however, might be widespread; for example, an organization's mainframe computer installation might be destroyed by fire.

Periodically, surveys have been undertaken of organizations to assess the adequacy of their disaster recovery plans. A common, recurring finding is that the quality of disaster recovery plans, if an organization even has one, is low. This situation exists even though other surveys report that on average the length of time organizations can survive in the event their information-processing function is lost is decreasing. Perhaps these findings reflect that disaster recovery plans are costly and difficult to prepare, maintain, and test. In organizations that have extensive decentralization and distribution of computing resources, for example, disaster recovery planning will be an onerous activity. For a start, security administrators are likely to have difficulty obtaining a commitment from large numbers of microcomputer users to maintaining effective backup.

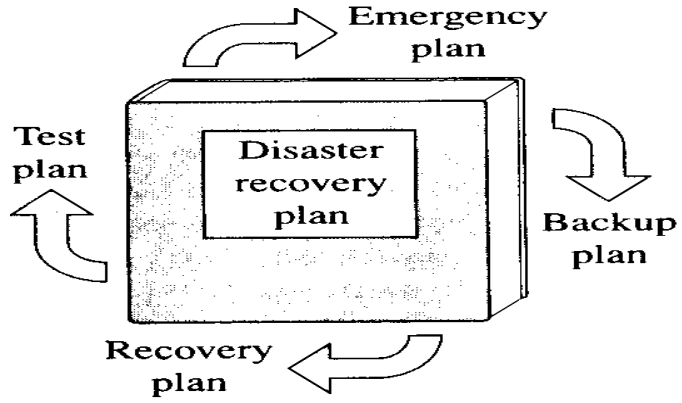


FIGURE 2-10 Disaster recovery plan and its components

Auditors are concerned to see that the organizations audited have appropriate, high-quality disaster recovery plans in place. Because the preparation, maintenance, and ongoing testing of disaster recovery plans

are often costly, the plan should be appropriate to the needs of the organization. Clearly, organizations that depend more on computers to support their operations will have greater needs. External auditors will be especially interested in a client's ability to continue as a going concern in the event disaster strikes and recovery cannot be affected quickly. They must also consider whether contingent claims might arise because contractual agreements the client has with other parties specify an appropriate, high-quality, regularly tested disaster recovery plan must be in place. Indeed, in some cases, clients might be governed by legislation that requires that they have appropriate, high-quality disaster recovery plans. Internal auditors will have the same concerns. In addition, they might also evaluate whether preparation, maintenance, and testing of the plan is carried out efficiently.

Comprehensive disaster recovery plan comprise four parts (1) an emergency plan, (2) a backup plan, (3) a recovery plan, and (4) a test plan (Figure 2-10). The plan lays down the policies, guidelines, and procedures for all personnel who have responsibility for the information systems function to follow. For example, it specifies the daily backup procedures that microcomputer users should follow and the site where recovery of mainframe operations is to be affected in the event of a fire. As a basis for assisting the audit evaluation, the following subsections briefly examine the nature, content, and preparation of each of the four parts of the plan.

2.4.2 Emergency Plan

The emergency plan specifies the actions to be undertaken immediately when a disaster occurs. Management must identify those situations that require the plan to be invoked for example, major fire, major structural damage, and terrorist attack. The actions to be initiated can vary somewhat depending on the nature of the disaster that occurs. For example, some disasters require that all personnel leave the information systems facilities immediately; others require a few select personnel remain behind for a short period to sound alarms, shut down equipment, and so on.

If an organization undertakes a comprehensive security review program, the threats identification and exposures analysis phases involve identifying those situations that require the emergency plan to be invoked. Each situation will be an exposure; that is, it will be a threat that eventuates and brings about losses because controls have failed or none exist to cover the threat.

2.4.3 Backup Plan

The backup plan specifies the type of backup to be kept, the frequency with which backup is to be undertaken, the procedures for making backup, the location of backup resources, the site where these resources can be assembled and operations restarted, the personnel who are responsible for gathering backup resources and restarting operations, the priorities to be assigned to recovering the various systems, and a time frame in which recovery of each system must be effected. For some resources, the procedures specified in the backup plan might be straightforward. For

example, microcomputer users might be admonished to make backup copies of critical files and store them off site. In other cases, the procedures specified in the backup plan could be complex and somewhat uncertain. For example, it might be difficult to specify exactly how an organization's mainframe facility will be recovered in the event of a fire.

The backup plan needs continuous updating as change occurs. For example, as personnel with key responsibilities in executing the plan leave the organization, the plan must be modified accordingly. Indeed, it is prudent to have more than one person knowledgeable in a backup task in case someone is injured when a disaster occurs. Similarly, lists of hardware and software must be updated to reflect acquisitions and disposals.

Perhaps the most difficult part in preparing a backup plan is to ensure that all critical resources are backed up. The following resources must be considered:

Resource	*Nature of Backup*
Personnel	Training and rotation of duties among information systems staff so they can take the place of others. Arrangements with another company for provision of staff.
Hardware	Arrangements with another company for provision of hardware.
Facilities	Arrangements with another company for provision of facilities.
Documentation	Inventory of documentation stored securely on site and off site.
Supplies	Inventory of critical supplies stored securely on site and off site with list of vendors who provide all supplies.
Data/information	Inventory of files stored securely on site and off site.
Applications software	Inventory of application software stored securely on site and off site.
Systems software	Inventory of systems software stored securely on site and off site.

The selection of backup sites is an important decision. These sites must be close enough to enable easy pickup and delivery of backup resources. They must be sufficiently distant, however, so it is unlikely that both the organization's information systems facilities and the backup-site facilities will be destroyed as the result of a single disaster. In some cases, this objective might be difficult to achieve. For example, if a major earthquake were to occur, nearby backup sites might also be destroyed.

Provision of suitable backup for mainframe computers usually is a more difficult task than provision of suitable backup for minicomputers and microcomputers. Replacement minicomputers and microcomputers often can be obtained quickly. Furthermore, usually they have minimum requirements in terms of an appropriate operating environment. Mainframe computers, on the other hand, typically require specialized operational facilities. Security administrators should consider the following backup options:

1. Cold site: If an organization can tolerate some downtime, cold-site backup might be appropriate. A cold site has all the facilities needed to install a mainframe system raised floors, air conditioning, power, communications lines, and so on. The mainframe is not present, however, and it must be provided by the organization wanting to use the cold site. An organization can establish its own cold-site facility or enter into an agreement with another organization to provide a cold-site facility.

2. Hot site: If fast recovery is critical, an organization might need hot-site backup. All hardware and operations facilities will be available at the hot site. In some cases, software, data, and supplies might also be stored there. Hot sites are expensive to maintain. They usually are shared with other organizations that have hot-site needs.

3. Warm site: A warm site provides an intermediate level of backup. It has all cold-site facilities plus hardware that might be difficult to obtain or install. For example, a warm site might contain selected peripheral equipment plus a small mainframe with sufficient power to handle critical applications in the short run.

4. Reciprocal agreement: Two or more organizations might agree to provide backup facilities to each other in the event of one suffering a disaster. This backup option is relatively cheap, but each participant must maintain sufficient capacity to operate another's critical systems. Reciprocal agreements are often informal in nature.

If a third-party site is to be used for backup and recovery purposes, security administrators must ensure that a contract is written to cover such issues as (1) how soon the site will be made available subsequent to a disaster, (2) the number of organizations that will be allowed to use the site concurrently in the event of a disaster, (3) the priority to be given to concurrent users of the site in the event of a common disaster, (4) the period during which the site can be used, (5) the conditions under which the site can be used, (6) the facilities and services the site provider agrees to make available, and (7) what controls will be in place and working at the off-site facility. These issues are often poorly specified in reciprocal agreements. Moreover, they can be difficult to enforce under a reciprocal agreement because of the informal nature of the agreement.

The need for backup highlights the value of using hardware and system software that conform to widely accepted standards and developing portable application systems. Specialized hardware and software might be more effective and more efficient, but they undermine an organization's ability to recover from a disaster quickly.

The recovery component of the backup plan needs careful consideration. In the event of a disaster, personnel will be responsible for tasks they undertake infrequently. Furthermore, they might be working under stress in an unfamiliar environment. The backup plan must assist them by providing concise, complete, clear instructions on recovery procedures they must follow.

2.4.4 Recovery Plan

Whereas the backup plan is intended to restore operations quickly so the information systems function can continue to service an organization, recovery plans set out procedures to restore full information systems capabilities. The specifics of how recovery is to be affected are often difficult to articulate. They depend on the circumstances of the disaster. For example, they will depend on whether the disaster is global or localized and, if localized, the nature of the machine (e.g., microcomputer, minicomputer, and mainframe), the applications, and the data to be recovered. In this light, recovery plans should identify a recovery committee that will be responsible for working out the specifics of the recovery to be undertaken. The plan should specify the responsibilities of the committee and provide guidelines on priorities to be followed. For example, certain members of the committee could be responsible for hardware replacement. The plan might also indicate which applications are to be recovered first.

2.4.5 Test Plan

The final component of a disaster recovery plan is a test plan. The purpose of the test plan is to identify deficiencies in the emergency, backup, or recovery plans or in the preparedness of an organization and its personnel in the event of a disaster. It must enable a range of disasters to be simulated and specify the criteria by which the emergency, backup, and recovery plans can be deemed satisfactory.

Periodically, test plans must be invoked; that is, a disaster must be simulated and information systems personnel required to follow backup and recovery procedures. Unfortunately, top managers are often unwilling to carry out a test because daily operations are disrupted: They also fear a real disaster could arise as a result of the test procedures.

To facilitate testing, a phased approach can be adopted. First, the disaster recovery plan can be tested by desk checking and inspection and walkthroughs, much like the validation procedures adopted for programs. Next, a disaster can be simulated at a convenient time for example, during a slow period in the day. Anyone who will be affected by the test (e.g., personnel and customers) also might be given prior notice of the test so they are prepared. Finally, disasters could be simulated without warning at any time. These are the acid tests of the organization's-ability to recover from a real catastrophe.

2.4.6 Insurance

Insurance sometimes can be used to mitigate losses that arise when disasters eventuate. Policies usually can be obtained to cover the following resources:

Insurance Area	*Explanation*
Equipment	Covers repair or acquisition of hardware. Varies depending on whether the equipment is purchased or leased.
Facilities	Covers items such as reconstruction of a computer room, raised floors, special furniture.
Storage media	Covers the replacement of the storage media plus their contents—data files, programs, documentation.
Business interruption	Covers loss in business income because an organization is unable to trade.
Extra expenses	Covers additional costs incurred because an organization is not operating from its normal facilities.
Valuable papers and records	Covers source documents, preprinted reports, documentation, and other valuable papers.
Accounts receivable	Covers cash-flow problems that arise because an organization cannot collect its accounts receivable promptly.
Media transportation	Covers damage to media in transit.
Malpractice, errors, and omissions	Covers claims against an organization by its customers, e.g., claims and omissions made by the clients of an outsourcing vendor or service bureau.

Insurance coverage for certain types of losses, however, might be difficult, if not impossible, to obtain. For example, security administrators might be unable to purchase insurance to mitigate losses from some types of computer crime, such as, destruction of critical files by a virus. Information systems insurance is still a reasonably new area, and the types of policies that are available are continuing to evolve.

When an insurance policy has been written, security administrators must ensure that their organizations fulfill any obligations under the policy. For example, some policies require the insured have an up-to-date, comprehensive disaster recovery plan. In addition, usually the insurer must be notified of any substantive change in risk. Just what constitutes a substantive change in risk might be unclear. Usually it includes changes to hardware, but the position with respect to changes to software and data files may be uncertain.

If organizations act as an outsourcing vendor or a service bureau or use an outsourcing vendor or a service bureau to perform some of their information systems functions, special care must be taken to establish responsibilities for safeguards as a basis for determining the types of insurance that are then needed. The following matters must be considered:

1. What liability does the outsourcing vendor or service bureau have for malpractice, errors, and omissions?

2. What liability does the outsourcing vendor or service bureau have for failure to deliver promised services?

3. Who owns each of the programs and data files maintained by the outsourcing vendor or service bureau?

4. What are the respective responsibilities of the outsourcing vendor or service bureau and customers with respect to backup and recovery?

Each of these matters needs to be addressed contractually. If possible, residual exposures should then be covered by insurance.

2.5 Some organizational issues

Depending on the size of an organization and its reliance on its information systems function, the security-administration role can occupy four possible positions within the organizational hierarchy (Figure 2-11). In small organizations that use turnkey systems that is, hardware and software systems that have been purchased as a package rather than developed in

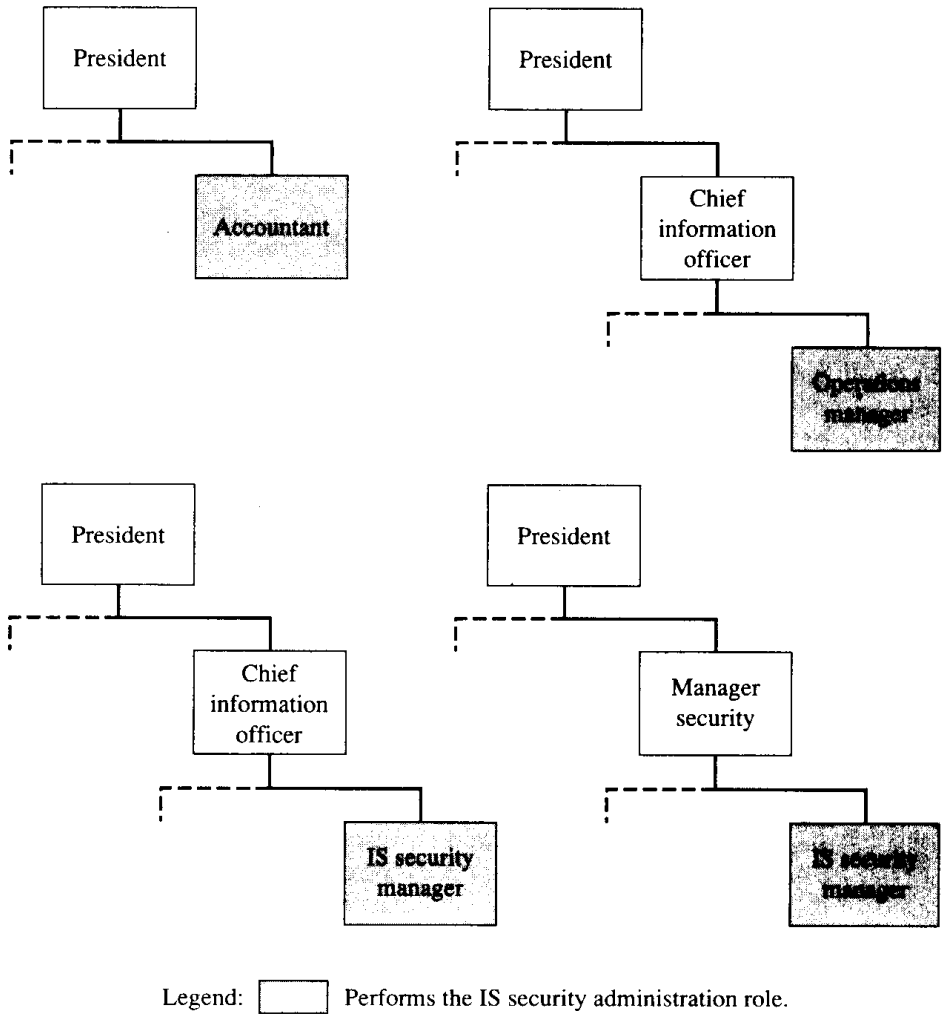


FIGURE 2-11. Organizational location of the IS security administration function.

house there might be no full-time professional information systems staff. As a consequence, no one obvious might be available to assume the security-administration role. Nonetheless, even in a small organization, security administration is still important. A small business that processes all its accounting records on a microcomputer could be forced into liquidation, for example, if a malicious employee or a virus were to destroy its critical master files. Someone, therefore, must take responsibility for security. Possibly the organization's president or accountant might assume this role.

If an organization has its own information systems staff but insufficient work to justify an ongoing, separate security administration position, responsibility for security matters might be vested in the operations manager. Because operations managers are responsible for the day-to-day running of hardware and software systems, security-administration functions seem a natural extension of their responsibilities. A concern with this approach, however, is that operations managers then have direct access to systems as well as responsibility for security over these systems. A dishonest or malicious operations manager could easily compromise security.

3. Quality assurance (QA) management

Structure

Objectives

After going through this lesson, you should be able to:

- understand how to implement QA Functions
- discuss about a Organizational considerations
- understand Relationship between quality assurance and auditing

3.1 Introduction

Quality assurance (QA) management is concerned with ensuring that (1) the information systems produced by the information systems function achieve certain quality goals and (2) development, implementation, operation, and maintenance of information systems comply with a set of quality standards. The QA function has come into existence because many organizations now recognize they can no longer compete effectively unless they emphasize quality throughout all their operations. Moreover, they also recognize the

need for independent review of the work done by information systems development, maintenance, and operations staff. Even with the best intentions, people cannot properly evaluate the quality of their own work. Independent, objective assessments are required.

Like the data administration function, the information systems QA function is relatively new. Auditors might not find it in all organizations. It is more likely to exist in those organizations where the information systems function is large and those that produce software where the quality of the software is a paramount requirement. Where a QA function does exist, however, it can have an important impact on the conduct of audit work. QA personnel are concerned with controls compliance in the same way that auditors have this focus. If auditors find a high-quality QA function in place, most likely greater reliance can be placed on controls, and the extent of substantive testing during the audit can be reduced.

In this lesson we first address the motivations for establishing an information systems QA role within organizations. Next we examine the various functions that QA personnel should perform. We then discuss some organizational issues that relate to where the QA function should be placed within the organizational hierarchy and how it should be staffed. Finally, we examine briefly how the existence of a QA function impacts the audit function.

3.2 Motivations toward the QA role

There are six reasons why the information systems QA role has emerged in many organizations (Figure 3-1). First, increasingly organizations are producing safety-critical information systems. For example, software is now used to support air traffic control, weapon guidance systems, ship guidance systems, nuclear reactors, radiation therapy equipment, heart pacemakers, and drug infusion systems. Errors in these systems can have devastating effects, such as loss of life, extensive destruction of property, and widespread damage to the environment. Organizations that produce safety-critical software must strive to ensure it is error free.

Second, users are becoming more demanding in terms of their expectations about the quality of software they employ to undertake their work. Many organizations that produce software now believe they will not be able to compete effectively in the marketplace unless their software meets stringent quality control standards. Otherwise their customers will simply switch to competitors who produce higher-quality software. Some customers also require that software developers meet certain quality standards. In Australia, for example, in the absence of mitigating circumstances, the Queensland State Government will not purchase software from developers unless they have been certified by a third party as meeting Australian Standard AS3563—Software Quality Management System.

Third, organizations are undertaking more ambitious projects when they build software. As information systems development skills, tools, and

methodologies improve and the needs of more straightforward applications have been addressed, organizations are looking to pioneering and innovative applications for software in an attempt to gain a competitive advantage. These applications are often large and complex. As a result, their development, implementation, operation, and maintenance are difficult. If these applications are to accomplish their objectives, they must meet stringent quality control standards.

Fourth, organizations are becoming increasingly concerned about their liabilities if they produce and sell defective software. Although the law relating to this area is still evolving, there are signs that companies might be liable for losses caused by defective software unless they have taken reasonable and prudent steps to ensure the quality of the software they market. In any event, even if eventually a company is not found liable for defective software it has produced and sold, the loss of customer goodwill that might arise could severely undermine its profitability.

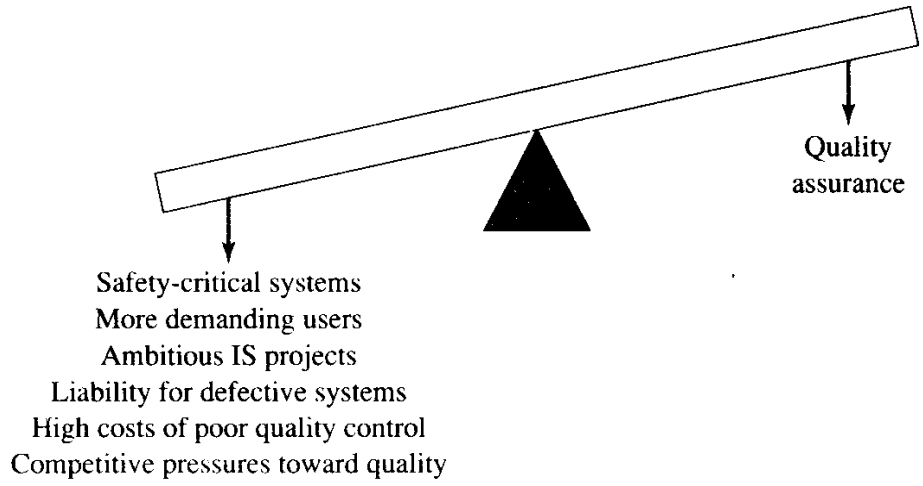


FIGURE 3-1 Motivations toward implementation of the information systems quality assurance function.

Fifth, poor quality control over the production, implementation, operation, and maintenance of software can be costly in terms of missed deadlines, dissatisfied users and customers, lower morale among information systems staff, higher maintenance, and strategic projects that must be abandoned. In some cases the senior management of organizations has had to explain to shareholders why software development debacles have severely affected profits, dividends, and share prices. In short, high-quality information systems and high profitability are inextricably bound in some organizations. Indeed, many proponents of quality management argue it is free. The benefits of quality management exceed its costs.

Sixth, improving the quality of information systems is part of a worldwide trend among organizations to improve the quality of the goods and services they sell. Many Western organizations, for example, are seeking to emulate

the success of Japanese companies which have had an unbending focus on customer needs, continuous quality improvement, and the production of zero-defect goods and services. Total quality management (TQM) is now a major driving force in many organizations because of their need to compete effectively in international marketplaces.

3.3 QA Functions

QA personnel should work closely with information systems personnel to improve the quality of information systems produced, implemented, operated, and maintained in an organization. They perform a monitoring role for management to ensure that (1) quality goals are established and understood clearly by all stakeholders and (2) compliance occurs with the standards that are in place to attain quality information systems. Like auditors, QA personnel should not assume responsibility for performing development, implementation, operations, or maintenance work. Otherwise, they will lose their independence. Moreover, they will become embroiled in costly, time-consuming disputes over specific issues and lose perspective on whether the information systems function overall is striving to meet quality goals.

3.3.1 Developing Quality Goals

One of the more difficult tasks that QA personnel must undertake is to develop quality goals for the information systems function and to develop or approve quality goals for specific information systems. Three problems arise. The first is that quality can have a different meaning depending on whose perspective is adopted. Top management, for example, might evaluate quality in terms of whether an information system allows their organization to compete better in a marketplace. Programmers, on the other hand, might evaluate quality in terms of whether the program code for an information system is well structured. Customers might evaluate quality in terms of the level of functionality the system provides. Many different notions of quality might have to be taken into account, therefore, when formulating quality goals.

The second problem is that' quality goals might need to vary across information systems. In safety-critical systems, for example, accuracy and completeness may be paramount objectives. In a strategic planning system, however, achieving timely reporting could override data accuracy and completeness goals. In short, quality goals can vary, depending on the nature of the system to be developed, operated, and maintained.

The third problem is that quality goals could be in conflict with one another. For example, one quality goal for an information system might be a fast response time. This goal might be achieved, however, only by writing complex algorithms that are difficult for programmers to understand and maintain. Specifying the goals for information systems, therefore, sometimes requires that decisions be made on how one goal will be traded off against another.

At the level of specific information systems, however, articulating quality goals is more difficult. If QA personnel are responsible for preparing the quality plan, they must work with the stakeholders in an information system to elicit the quality goals they deem important. These goals can differ across information systems. For example, as discussed previously, in some cases the goals of an information system may focus on accuracy and completeness; in other cases timeliness goals may mean that accuracy and completeness goals have to be compromised.

Table 3-1 Software quality characteristics

Quality Characteristic	*Explanation*
Functionality	Extent to which the software contains the functions needed tc satisfy user needs.
Reliability	Extent to which the software sustains its level of performance under stated conditions for some defined time period.
Usability	Level of effort needed for users to exploit the functionality of the software.
Efficiency	Level of resources consumed by the software to perform its functions.
Maintainability	Level of effort needed to modify the software.
Portability	Extent to which software can be transferred from one hardware/software platform to another.

Table 3-1 shows six quality characteristics that the International Organization for Standardization has adopted in their standard for Software Product Evaluation (ISO 9126, 1991). QA personnel can use these characteristics to articulate quality goals for *individual* information systems. The six characteristics provide a checklist that QA personnel can use during their interviews with stakeholders to determine quality goals. The levels required for each characteristic can be evaluated, and trade-offs that need to be made among the characteristics can be considered.

When formulating or evaluating the quality goals for a specific information system, QA personnel must take care to ensure the specific goals are not in conflict with the overall goals of the organization. For example, to achieve certain strategic goals for the organization, management might require that information systems not be sold to customers if the likelihood of a system containing a defect is above a certain level. It might be impossible to achieve this goal, however, if a particular system is to be developed within a time frame that is acceptable to its customers. Either management must relinquish the overall quality goal, or customers must compromise with respect to the development time frame. If neither of these alternatives is acceptable, the organization may have to desist from developing the system.

Similarly, the goals of individual information systems can be in conflict. For example, to achieve the desired response time for an information system, substantial operational resources (e.g., the amount of memory and disk space) might have to be allocated to the system. As a result, response times

for other systems that operate concurrently with the system may decline to unacceptable levels. QA personnel must be mindful of quality goals for a specific information system in the context of quality goals that exist for other information systems.

Gaining acceptance among stakeholders of the quality goals for a specific information system can be difficult. In the case of system-development staff, for example, they might consider that quality goals place unwanted constraints on the ways they work. Only token acceptance of quality goals might occur. As a result, quality goals are compromised quickly when problems are encountered during system development. For this reason, some organizations require that the quality plan for a specific information system be prepared by a quality-control group within the project team. These organizations believe quality goals are more likely to be *owned* by the project team if team members have responsibility for preparing the quality plan. QA personnel then play the role of approving the quality plan prepared by the project team and monitoring how the project team uses the plan to achieve quality goals (Figure 3-2).

When quality goals have been determined for a specific information system, it is important that quality metrics be chosen. Quality goals are unlikely to be accomplished satisfactorily unless information systems personnel know the criteria to be used to determine whether quality goals have been met. Because the adequacy of various quality measures can be a contentious area, it is important that QA personnel again consult all stakeholders and try to negotiate agreement on the quality metrics to be used.

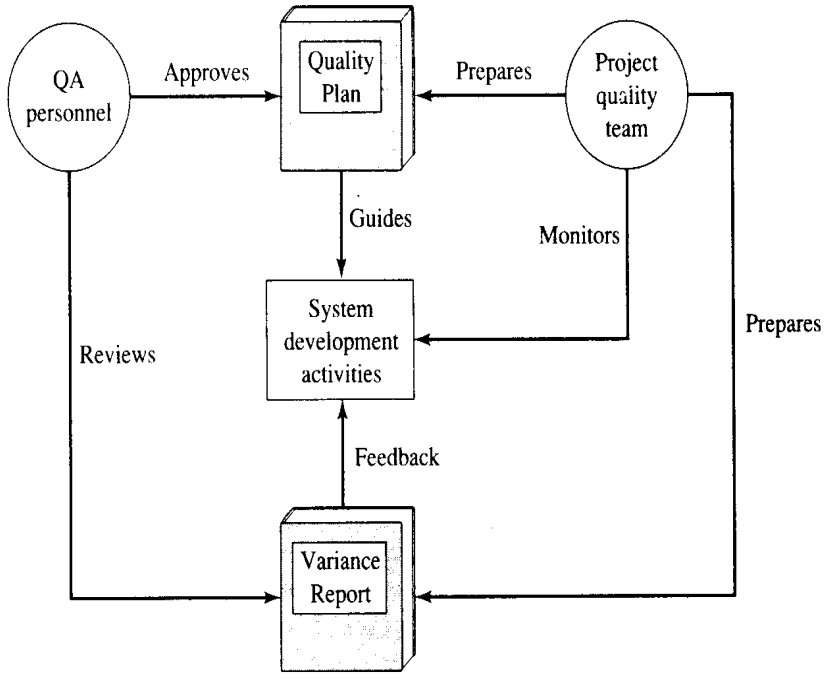


FIGURE 3-2 using a project quality plan to attain quality goals

3.3.2 Developing, Promulgating, and Maintaining Standards for the Information Systems Function

Information systems standards are an important means of achieving asset safeguarding, data integrity, system effectiveness, and system efficiency objectives. They must cover all major tasks performed within the information systems function for example, requirements analysis, design, programming, testing, documentation, operations, and maintenance. They must also recognize that substantial diversity exists in the ways the information systems function is now performed. For example, they must take into account that information systems are developed under varying levels of uncertainty, that information systems functions are sometimes performed on mainframe computers and sometimes performed on microcomputers, and that information systems are sometimes developed by information systems professionals and sometimes by end users. In general, the objective is to provide *minimal specification of standards:* The standards should enable quality objectives to be attained, but they should not stultify the ways in which information systems personnel must work to accomplish their jobs.

There are four advantages to having QA personnel assume responsibility for the development, promulgation, and maintenance of information systems standards. First, QA personnel are charged with being knowledgeable about and remaining up to date with best practice in information systems standards. They should be familiar with the types of standards that have been adopted by organizations that are similar to their own. Furthermore, they should be knowledgeable about standards that are being developed or those that have been adopted by national or international standards organizations. Given the substantial work that is now being undertaken worldwide on information systems standards, organizations must make a major commitment to keep up to date with developments and to understand the potential impacts that standards may have on their own activities.

Second, within an organization, decisions on standards can often be a political issue that evokes strong, emotive reactions. Standards can have a major impact on the ways work is done. Some people perceive that standards will inhibit their work; others will argue that standards allow too much freedom; still others will be concerned about how standards affect the formal or informal power they hold. Consequently, many people have a vested interest in the outcome of standards decisions. Of all stakeholders, QA personnel are likely to be perceived as the most independent if they assume responsibility for the development, promulgation, and maintenance of information systems standards. QA personnel are charged with adopting an organization wide view on standards. To the extent they do not adopt this broad perspective, their position within the organizations is likely to become untenable.

Third, QA personnel are supposed to undertake analyses of the reasons when an organization fails to achieve its information systems quality goals (see the following section). In this light, they should obtain insights and understanding that will allow them to make good judgments on the

characteristics of standards that are best suited to their organizations. They should be able to identify any new standards that need to be developed or any existing standards that need to be modified based upon the follow-up analyses of information systems problems that they undertake. Again, they are likely to be better placed than any other stakeholder to make judgments on standards because their work allows them to obtain a broad overview of information systems activities within their organizations.

Fourth, QA personnel should have incentives to ensure that their organization adopts and complies with the best set of information systems standards possible. No other personnel within the organization are likely to have their performance evaluated on the basis of attainment of quality goals to the extent that QA personnel are evaluated on this basis. Moreover, other personnel inevitably face dilemmas because quality goals conflict with other goals that are more central to their role. For example, development personnel might face a situation in which deadlines will be missed if they comply with documentation standards. If senior management establishes the correct incentives for the QA function, it should always be in the best interests of QA personnel to seek to achieve both the short- and long-term quality goals of the organization.

QA personnel have an important role to play in *monitoring* national and international information systems standards that could affect their organization. These standards should inform the process of developing the overall standards to be used by an organization's information systems function and any specific standards that are needed to support the work conducted in relation to a particular information system project (Figure 3-3).

QA personnel also might want to actively *participate* in the development of national or international standards. In some cases, these standards can be exceedingly helpful to an organization attaining its goals. For example, agreement on some type of data communication standard can allow an organization to interact well electronically with its customers and to strengthen its position within the marketplace. In other cases, however, standards could undermine an organization's competitive position. For example, the organization may have invested heavily in systems that are at odds with the standards eventually adopted. By participating in the standard-setting process, QA personnel can seek to safeguard the interests of their organizations and provide timely warning when adverse standards are likely to be adopted.

QA personnel also have an important role to play in monitoring *best practices* in other organizations. As with national and international standards, best practices can affect the specific information system processes that are adopted and enforced as standards within an organization. In some cases, knowledge about best practice can be obtained via publications. For example, the *clean-room method* of software development is widely accepted as best practice in producing software with known and certified mean time to failure (MTTF). It involves relying heavily on software practitioners using formal specifications, formal verification, formal design

and code inspections, independent software product testing, and statistical process control. The processes associated with the clean room method are now well documented. In other cases, however, knowledge about best practice is not readily available. It can be obtained only by QA personnel fostering a network of professional contacts and attending conferences and professional meetings.

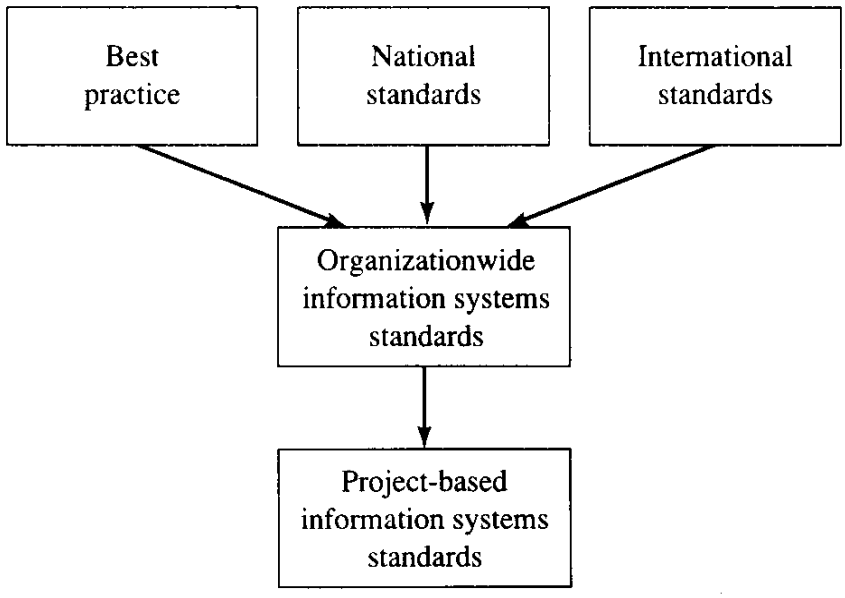


FIGURE 3-3 informing the standards development process.

Whatever standards or practices are chosen, QA personnel must take care to match the capabilities of their organization with the demands that arise from seeking to adhere to different types of standards or practices. In this regard, the Software Engineering Institute at Carnegie Mellon University has developed a Capability Maturity Model (CMM) that defines five levels of organizational maturity and the software quality processes associated with each of these levels (Figure 3-4). Organizations are admonished not to try to jump levels. Rather, they are urged to undertake continuous process improvement and to move through these levels in an evolutionary way, Only by mastering the software quality processes associated with lower levels in the CMM is an organization likely to be able to implement successfully the software quality processes associated with higher levels in the CMM.

Auditors can use interviews, observations, and reviews of documentation to evaluate how well QA personnel develop, promulgate, and maintain standards for the information systems function. They can ask QA personnel about the procedures they use to develop, promulgate, and maintain standards. Similarly, they can ask stakeholders to evaluate how well QA

personnel undertake these activities. By attending meetings where QA personnel focus on standards, auditors can observe how they undertake standards work. Because standards should be documented, they can review this documentation to assess the quality of the standards work carried out by QA personnel.

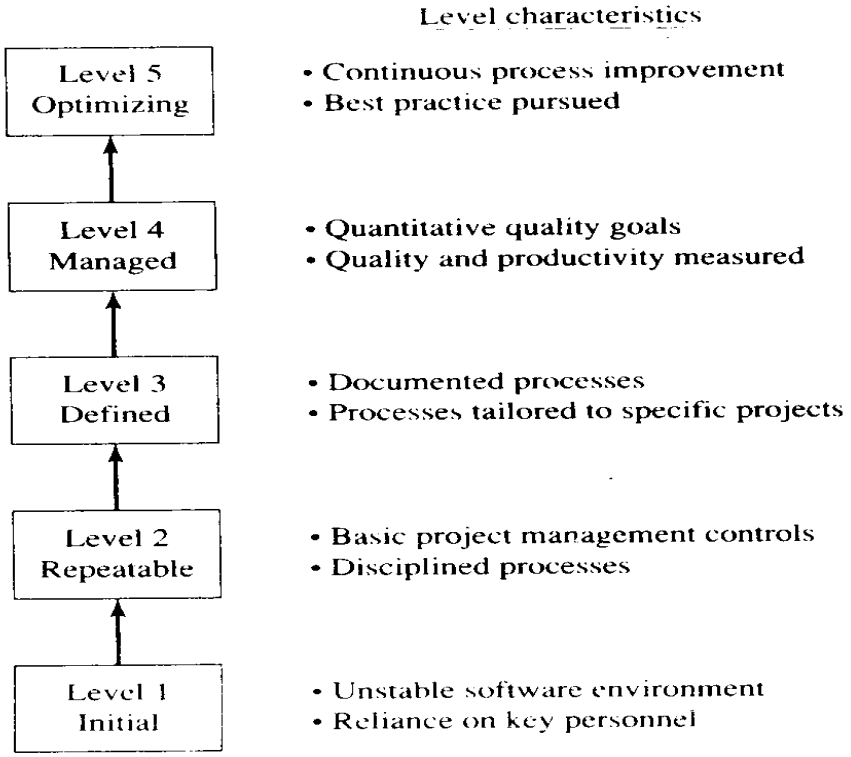


FIGURE 3-4 Characteristics of levels in Software Engineering Institute's capability maturity model

3.3.3 Monitoring Compliance with QA Standards

QA personnel undertake two types of monitoring of compliance with QA standards. First, they monitor compliance with the QA plan prepared for a *specific* system. In this regard they focus on development, implementation, operations, and maintenance activities associated with that system. In their compliance monitoring role, they might participate as moderators during design and code inspections, evaluate whether test data has been properly documented, and participate in review meetings when important milestones occur. They should also check to see that project personnel are themselves monitoring compliance with the quality plan prepared for the project. In this regard, they should see project personnel actively using standard quality-control tools that allow deviations from plans to be identified and improvements in project activities to be effected. For example, fishbone diagrams should be used to diagnose the causes of problems and to identify ways to mitigate these problems in future software projects (Figure 3-5).

QA personnel also should monkor compliance with *general* standards (standards that do not focus on a specific system). For example, an organization might have standards relating to the amount of ongoing professional development training that information systems staff must undertake. Certain managers might be responsible for ensuring that staff under take the requisite training. QA personnel, in turn, should evaluate whether both staff and management are complying with the training standard.

Two principles govern how compliance monitoring should be undertaken by QA personnel. First, they must remember that their role is to facilitate rather than to inhibit information systems development, implementation, operations, and maintenance. In this light they must be constructive and positive in their monitoring role. If they adopt an arbitrary, antagonistic, negative stance, they will lose credibility, and they no longer will be able to play an effective role.

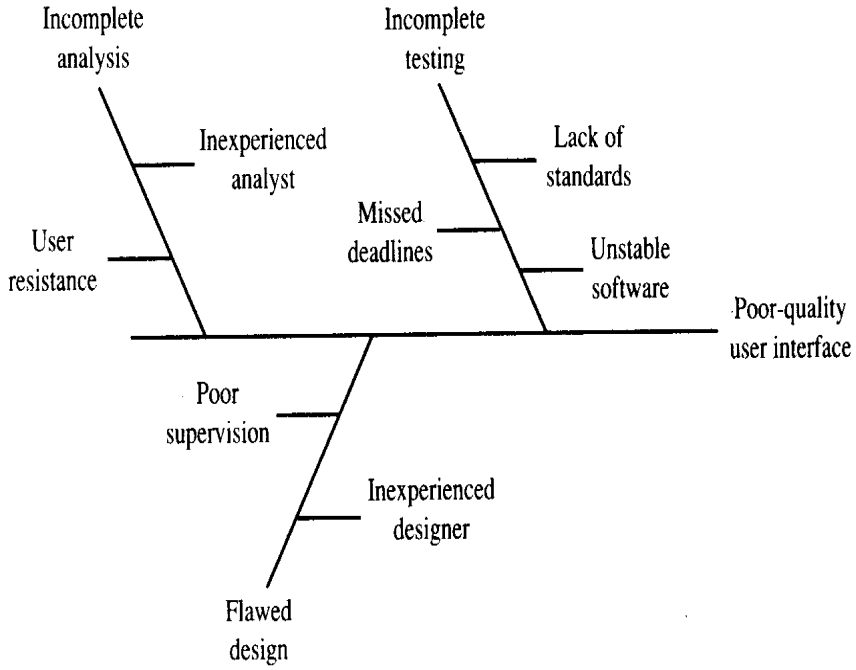


FIGURE 3-5 Fishbone diagram to analyze causes of a poor-quality user interface.

Second, QA personnel should seek to avoid disputes over detail. Instead, their role should be to notify management when compliance with standards has not occurred. Management is then responsible for resolving specific difficulties that have ensued or may ensue. Ultimately, management must decide whether it wishes to insist that compliance with standards occurs. If QA personnel engage in disputes over details and petitions to enforce compliance, they will lose the confidence of information systems staff and management.

When a compliance failure occurs, QA personnel should seek to understand the reasons for the failure so they can advise management. In some cases, there could be good reasons for noncompliance. Such reasons could motivate reconsideration of standards and ultimately bring about improved standards. In other cases, noncompliance could reflect a breakdown in a process, in which case corrective action will be needed.

QA personnel should also consider the consequences of compliance failure and brief management on these consequences. The repercussions could be serious if, for example, the failure results in a breach of contract with a customer or the potential arise for a substantial liability associated with a defective software product. On the other hand, the consequences might not be serious, at least in the short run, and urgent action might not be required.

When compliance failure occurs, QA personnel should consider appropriate corrective actions so they can make recommendations to management. Better recommendations can be made if QA personnel have a sound understanding of the reasons for the failure and the potential consequences of the failure. The recommendations are likely to be more concrete, more specific, and more compelling. Whether management acts on the recommendations, however, is their prerogative. Again, QA personnel should seek to avoid becoming involved in specific debates.

Compliance monitoring can be an important element of any certification that the organization has been given with respect to software quality. When the organization is being recertified, the certifying party might look to see that monitoring is carried out rigorously and that management takes appropriate actions in light of advice received on the basis of the monitoring process.

Auditors can use interviews, observations, and reviews of documentation to evaluate how well QA personnel perform their monitoring role. They can ask QA personnel what monitoring procedures they follow, talk with stakeholders to determine their experience of the QA monitoring process, observe QA personnel as they undertake the monitoring process, and review reports produced as a result of the monitoring process. Auditors should also seek to determine what actions have been undertaken as a result of the monitoring process to reach a decision on whether it is effective.

3.3.4 Identifying Areas for Improvement

An important outcome of QA personnel's monitoring role is that they should identify areas where the activities of the information systems function can be improved. As discussed previously, noncompliance with quality standards means either the standards have to be revised because they are deficient or a development; implementation, operations, or maintenance process has to be improved because there has been a genuine breakdown. In either case, non-compliance should lead to improved standards or processes.

For two reasons, QA personnel should have responsibility for identifying areas where the information systems function can be improved. First, given their overarching concern with quality assurance, they are in the best

position to offer *independent* advice. Other stakeholders, such as programmers and users, have vested interests. As a result, their recommendations will tend to lack credibility. For example, to reduce the number of missed development deadlines, programmers might argue that users have to be more diligent when they provide requirements specifications. Given the conflicts of interest that often exist between users and programmers, however, users are likely to be skeptical about such views. QA personnel, on the other hand, have no allegiance to a particular stakeholder group. Thus, their recommendations are more likely to be accepted. Nevertheless, they must be mindful that they will be given this trust only if they both act independently and are perceived to act independently. Like auditors, independence "in fact" and "in mind" are essential to the effective discharge of their duties.

Second, QA personnel should have the knowledge and experience to make the best recommendations for improvements to information systems standards or processes. QA personnel are supposed to keep up-to-date with the types of difficulties that all stakeholders in the information systems function encounter. Moreover, they should be aware of the best practice to overcome or to mitigate these difficulties. Given their duties, they also should have a broad appreciation of activities across all stakeholder groups. In this light, they should be able to address issues such as *(1)* the likelihood that information systems activities can be improved cost effectively, *(2)* the appropriateness of changes to information systems standards, *(3)* the impact that changes to standards or processes will have on all stakeholder groups, *(4)* the likelihood that changes to standards or processes will evoke behavioral resistance, and *(5)* the support that is likely be forthcoming from management if standards or processes are changed.

As discussed previously, regular monitoring for compliance of information systems activities against standards provides an important basis for QA personnel identifying areas where the information system function can be improved. In addition, material information systems *errors that* occur should also be reported routinely to QA personnel. For example, QA personnel should be informed when an application program updates a database incorrectly or an application program aborts prematurely during production execution. An error-management system should be in place to record these types of errors and to report them to QA personnel on a timely basis. QA personnel should then analyze the errors to identify their causes and to determine whether standards or processes need to be improved. For example, in the short run, QA personnel might be able to alert programmers that certain types of errors have been made frequently in programs that update distributed databases. In the long run, program testing standards may be modified to detect these errors before programs are released into production.

As with recommendations for corrective actions when noncompliance with standards occurs, recommendations for improved standards or processes will be more compelling if they are based on analyses of particular deficiencies that have occurred, a sound understanding of the reasons for

these deficiencies, and careful argumentation as to why the recommendations will overcome or mitigate the deficiencies. Stakeholders, especially management, are unlikely to be convinced by abstract arguments. Consequently, QA personnel should strive to be specific and concrete in the recommendations they make.

When conducting activities to identify areas for improvement, QA personnel should be guided by the Japanese notion of *Kaizen*. *Kaizen* means "ongoing improvement." It involves everyone in an organization (managers and workers), and it has been fundamental to Japan's success with TQM. It stresses the need for a continuous, process-oriented cycle of planning, approval; execution, monitoring, and evaluation (Figure 3-6). In this light, QA personnel must regard the identification of areas for improvement as an ongoing activity rather than a periodic activity. Furthermore, *Kaizen* focuses on the need to base improvement decisions on *facts* rather than experience or intuition. QA personnel should seek, therefore, to make recommendations for improvement only if they have a substantive empirical basis to back up their recommendations.

Auditors can evaluate how well QA personnel make recommendations for improved standards or processes through interviews, observations, and reviews of documentation. For example, they can interview stakeholders in the information systems function to obtain their opinions on the quality of advice provided by QA personnel with respect to improvement in standards and processes. They can observe the procedures followed to report material errors to QA personnel and the actions they follow upon being notified of material errors. Auditors can review reports to assess the quality of the recommendations provided by QA personnel with respect to improvements in standards and processes. In particular, auditors should seek to determine what actions were taken by management in light of the recommendations. If they conclude that QA personnel function effectively to provide advice on improved standards and processes, they can have greater confidence in the overall reliability of information systems controls. Accordingly, it is likely auditors will be able to reduce the extent of substantive testing. If they do not function effectively, however, most likely the extent of substantive testing will have to be increased.

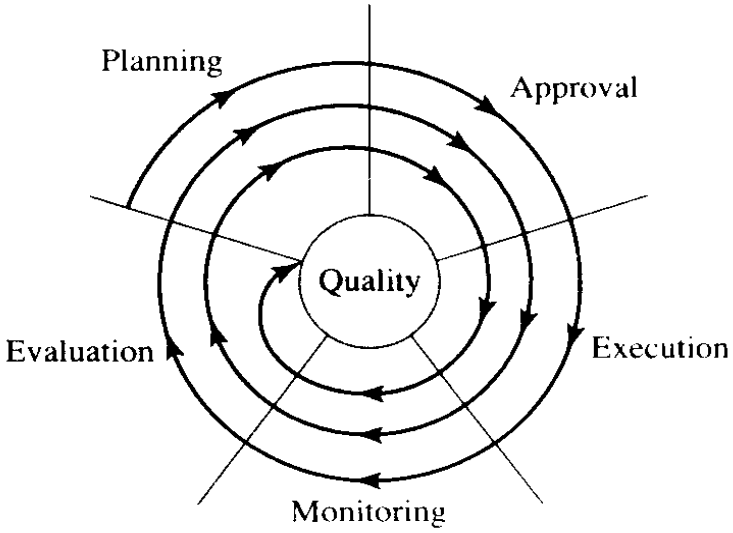


FIGURE 3-6 the quality spiral.

3.3.5 Reporting to Management

QA reporting is an important but difficult undertaking. Given the nature of the QA function, stakeholders could be quick to take umbrage with the contents of reports. As a result, political difficulties can ensue. Moreover, it is sometimes difficult to identify who should receive a QA report in order to achieve the greatest impact.

QA personnel must take great care in preparing their reports. The reports must not degenerate into a long list of defects that have been identified. Otherwise, they will cause anger among stakeholders, and conflict will arise. Instead, the reports must focus on major findings and fundamental issues that need to be addressed. Recommendations must be backed up by sound analyses based on concrete facts. The reports also must be positive in nature, and they should contain no surprises. Otherwise the reports will evoke protracted escalation meetings where stakeholders seek to defend their positions. When this outcome arises, inevitably QA personnel lose their credibility, and the effectiveness of the QA function is undermined.

Regular reports on compliance with *general* information systems standards should be provided to the manager of the information systems function. Those who should be the recipients *of project-based* reports, however, are less clear-cut. Project managers need to receive copies of these reports because they provide the basis for better management of their projects. Moreover, they have a right to know the contents of reports about projects under their control. Project managers have vested interests in projects under their control, however, and as a result they may resist recommendations for change. In this light, QA reports must be provided to stakeholders who have

opposing interests. The views of one group who oppose the changes can then be counterbalanced by another group who support the changes. Nevertheless, lines of reporting must be agreed on at the outset of a project so recipients are not altered midstream to further some group of stakeholder's political ends.

Auditors can evaluate how well QA personnel undertake the reporting function through interviews, observations, and reviews of documentation. For example, auditors can interview QA personnel to determine the approaches they use to report their findings. They can also interview stakeholders to determine their levels of satisfaction with the QA reporting process. When meetings are called to discuss the contents of reports, auditors can attend to see how the reports are received and what changes they evoke. They can also review a sample of QA reports to obtain evidence to make a judgment on their quality.

If auditors conclude that QA personnel are carrying out the reporting function effectively, they can have greater confidence in the likelihood that QA efforts are having a material impact on the quality of information systems that are being developed, implemented, and maintained. Accordingly, auditors can reduce the extent of substantive testing. If, on the other hand, the reporting function is not being carried out effectively, QA activities are likely to have lit-tie impact on information systems practices. Even if all other QA functions are well carried out, poor reporting can negate their impact. As a result, auditors will have to increase the extent of substantive testing.

3.3.6 Training in QA Standards and Procedures

Training is an essential element to maintenance of and compliance with QA standards and procedures. Indeed, Deutsch and Willis argue that "the cornerstone of quality is skilled and motivated people." Without training, QA standards and procedures are likely to fall into disuse because they are not updated or due to either ignorance or apathy on the part of stakeholders.

QA personnel have responsibility for training all stakeholders in the infor-mation systems function in QA standards and procedures. They must under-take two types of training. The first focuses on providing *general* knowledge about quality assurance. It addresses the nature of and motivations for quality assurance. It also examines the general standards and procedures established to attain QA objectives. For example, stakeholders would be apprised of systems-development milestones that require formal approval and sign-off and requirements for user documentation that must be met before software will be released for production use.

The second type of training focuses on standards and procedures that will be *specific* to an application system. For example, because a particular application to be developed will support life-critical activities, a third party might be employed to undertake independent verification and validation of the system. In the normal course of events, an organization might not develop life-critical applications. As a result, specific QA standards and procedures might have to be formulated for the application. These specific

standards and procedures should be identified when the QA plan for the application is prepared. Information systems personnel associated with the development, operation, and maintenance of the system then must be appraised of these standards and procedures.

QA training should be focused. A personal development plan should exist for each information systems employee. This plan should reflect both the organization's and the employee's goals and a development strategy for the employee to attain these goals. In relation to quality assurance, it should specify general education that the employee should acquire, specific training that the employee should undertake, and work experience that the employee should obtain. By analyzing the needs manifested in these personal development plans, QA personnel can formulate an overall QA training program for their organization.

QA training should also be ongoing. New employees must be inducted in the QA goals, standards, and procedures that have been adopted by the information systems function. Such induction is especially important when new employees are experienced information systems professionals who have *not* worked previously in organizations where quality is a paramount goal. Breaking undesirable work habits among experienced professionals can be a difficult problem to overcome. Subsequent training must he undertaken regularly. If a QA program is to work effectively, all stakeholders must be constantly reminded of its importance. Moreover, through ^regular training they can be admonished to achieve QA goals and objectives and reminded how they can achieve these goals and objectives.

QA personnel can use in-service training as an important means of discernment in relation to QA standards and procedures. Stakeholders have an opportunity to articulate the problems they have experienced in complying with QA standards and procedures. Discussion and debate can ensue. Furthermore, the shared wisdom of the group can be brought to bear to help resolve problems that people might be experiencing and to provide feedback on ways in which QA standards and procedures can be improved. In short, QA personnel should actively use in-service training opportunities to enact two-way communication: from themselves to stakeholders, and from stakeholders to themselves (Figure 3-7).

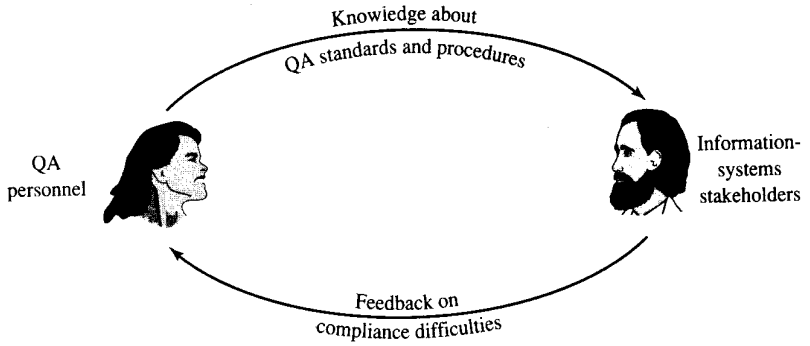


FIGURE 3-7 Quality assurance in-service training as a discernment process.

The quality of QA training is an important indicator of management's level of commitment to the production of quality information systems products and services. In the absence of management providing adequate resources for training, stakeholders are likely to become skeptical about management's commitment to quality. Quality goals will not be accomplished if stakeholders are not provided with the knowledge and experience to attain them.

Auditors can evaluate how well QA personnel manage training by using interviews, observations, and reviews of documentation. They can interview QA personnel to obtain an understanding of the approach they follow to undertake QA training within their organization. Another option is to interview stakeholders to obtain their views on whether the nature, scope, and frequency of QA training are satisfactory. By observing QA training sessions in progress, auditors obtain evidence on how well they are conducted. Reviews of training schedules and training materials also provide evidence on whether QA training is being carried out satisfactorily. Moreover, auditors can examine personnel records to determine whether QA training is being properly directed to meet the needs of individual stakeholders.

If auditors conclude that QA personnel are carrying out the training function reliably, they can reduce the extent of substantive testing. If they conclude that training is not being carried out reliably, however, in the absence of compensating controls substantive testing must be expanded. Inadequate training usually means breakdowns in other areas. For example, monitoring for compliance with QA standards and procedures becomes problematic when stakeholders are not fully apprised of the standards and procedures in the first place. In short, the quality of training is likely to be an important indicator of how well QA personnel are carrying out other QA functions. It is also an important indicator of management's commitment to the QA function.

3.4 Organizational considerations

If the QA function is to be effective, it must be properly established within the organizational hierarchy of the information systems function. Moreover, it must be staffed properly.

3.4.1 Placement of the QA Function

The QA function must be placed within the organizational hierarchy of the information systems function so that it can act independently of other information systems activities. In this light, it cannot report to a manager responsible for development, maintenance, or operations. Otherwise, it will not be perceived as independent, and its activities also will be too easily subject to duress. Instead, it must report directly to the executive who has overall responsibility for the information systems function (Figure 3-8). Moreover, managers responsible for the QA function must be given appropriate seniority to enable them to maintain their independence and to ensure their arguments carry sufficient weight in any disputes that arise.

To operate effectively, the QA function must also have a properly approved charter. This charter should be prepared in consultation with information systems stakeholders. When the final form of the charter has been negotiated, it should be endorsed by senior management so it has force in terms of the ways information systems personnel conduct their activities. To avoid confusion and disputation, the charter should lay out clearly the rights and responsibilities of the QA function. Job positions must also be defined to fulfill responsibilities under the charter. Authority and accountability commensurate with the activities that have to be performed in each job should be specified. If the charter is to continue to have force, senior management will need to affirm periodically the importance of the charter. Moreover, they will have to provide adequate resources so the QA function can discharge its responsibilities under the charter.

Auditors must evaluate whether the QA function has been placed within the organizational hierarchy of the information systems function such that it can perform an independent evaluative role. They should determine that it reports directly to the executive in charge of the information systems function. Moreover, they should determine that a properly approved charter exists that documents the rights and responsibilities of the QA personnel. The charter should be supplemented by job descriptions that allow the charter to be fulfilled. Auditors can interview QA staff, information systems staff, and information systems users to determine the scope and depth of QA work and to assess whether funding of the QA function is adequate. They can also review documentation to evaluate whether the work that has been undertaken by QA personnel is consistent with their charter.

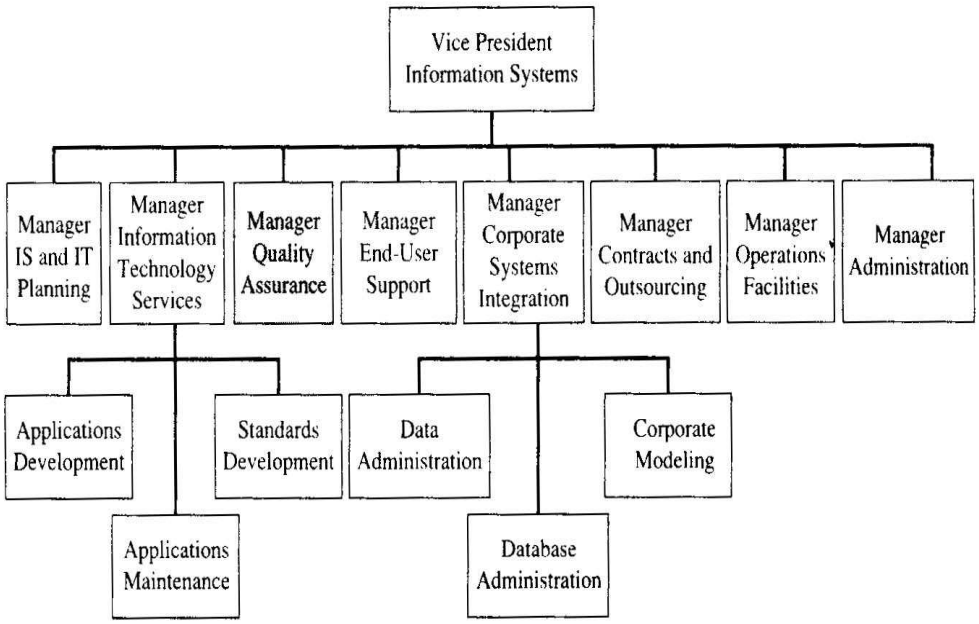


FIGURE 3-8 Placement of the QA function within the organizational hierarchy of the information systems function.

3.4.2 Staffing the QA Function

The QA function poses some special problems in terms of staffing. QA personnel must be well trained and competent, and their skills must be kept up to date. Otherwise, they will not command the respect of the information systems personnel whose work they must evaluate.

QA personnel also require a high level of interpersonal skills because the potential for conflict between QA personnel and information systems personnel is high. If interactions are not handled tactfully, the level of disputation is likely to become pathological. Senior management should have to become involved in disputes only on an infrequent basis. When they do, often the pressures to finish a job mean that senior management will support the development staff in a dispute rather than the QA staff. Long-term goals can be compromised to meet short-term objectives.

Many organizations report difficulties in attracting competent staff to QA positions. In general, information systems professionals prefer to develop, implement, and operate systems rather than to evaluate them for quality. Indeed, quality assurance work often is afforded lower prestige than other types of information systems work. If QA responsibilities are to be discharged effectively, however, competent, experienced personnel must be attracted to QA positions. Senior management must engender a culture that sees the QA role as important.

Auditors should evaluate whether QA personnel have adequate knowledge of information systems development, implementation, and operations procedures. They should also evaluate whether they receive ongoing, regular training. Through interviews with information systems users and information systems staff, auditors can determine whether QA staff is considered to be competent. They can also use these interviews to determine whether QA staff exercise a high level of interpersonal skills. If auditors conclude that QA staff lack required knowledge or their level of interpersonal skills is not high, they must place less reliance on QA controls when planning the remainder of the audit.

3.5 Relationship between quality assurance and auditing

In many ways, the objectives of and functions performed by QA personnel and auditors are the same. Both QA personnel and auditors are concerned with the existence of information systems standards, compliance with these standards, and timely, corrective actions when deviations from standards occur. Prior to the existence of the QA function, auditors performed many of the checks that QA personnel now perform. For example, they sought evidence on whether systems-development practices followed prescribed standards.

The existence of a QA function might change how both internal auditors and external auditors perform their work in several ways. First, if a QA function is

in place and working reliably, auditors most likely can reduce the extent of substantive testing they undertake. The existence of a QA function manifests an organization's commitment to quality. Moreover, the functions performed by QA personnel provide greater assurance that high-quality controls are in place and working over the information systems function. As a result, auditors can place greater reliance on controls during the conduct of their audits.

Second, QA personnel most likely will undertake more comprehensive checking of information systems controls than auditors. Although both QA personnel and auditors inevitably rely on sampling for their evidence-collection purposes, QA personnel probably will examine more extensive samples. As a .result, auditors usually can place higher reliance on information systems controls and reduce the extent of their substantive testing.

Third, auditors can now focus primarily on ensuring the QA function works reliably rather than undertaking direct tests of information systems controls. Providing auditors have confidence in the QA function, they can rely on the conclusions reached by QA personnel in relation to information systems controls and plan the conduct of their audit accordingly. Auditors should have less work to undertake because QA personnel should be able to provide them with much of the evidence they need to plan an audit and to reach an audit opinion.

In short, auditors should welcome the existence of a QA function in any organization. The control risk associated with the organization should be lower, and less work should be required to gather the evidential matter needed to reach an audit opinion. With proper planning, auditors can work hand in hand with QA personnel to achieve the four objectives of asset safeguarding, maintenance of data integrity, system effectiveness, and system efficiency.

4. SUMMARY

Program development and acquisition is a major phase within the systems development life cycle. The primary objective of this phase is to produce or acquire and to implement high-quality programs. The activities to be conducted during this phase can be managed in the context of a program development life cycle that comprises six phases: (1) planning, (2) control, (3) design, (4) coding, (5) testing, and (6) operation and maintenance. Auditors can use this life-cycle model to determine what types of activities should be conducted in each phase and to collect evidence on and to evaluate the conduct of these activities. Auditors must recognize that the ways in which these activities will be performed will vary, depending on such contingencies as the size of and complexity of the program to be developed. In this light auditors must be able to adjust the audit approach to take into account the effects of these contingent factors.

Information systems security administrators are responsible for ensuring that information systems assets are secure. Assets are secure when the expected losses that will occur from threats eventuating over some time period are at an acceptable level.

Quality assurance personnel within the information systems function are concerned with ensuring that the information systems produced achieve certain quality goals and that development, implementation, operation, and maintenance of information systems comply with a set of quality standards.

5. Questions

1. Give three attributes of a high-quality program.
2. What are the major phases in the program development life cycle? Which phase is a phantom phase? Briefly explain why it is a phantom phase.
3. Briefly describe two concerns auditors might have when evaluating the planning phase of the program development life cycle.
4. Briefly describe the two major purposes of the control phase of the program development life cycle.
5. How do techniques like Work Breakdown Structures, Gantt charts, and PERT charts help during the control phase of the program development life cycle?
6. Briefly explain the difference between unit testing, integration testing, and whole-of-program testing.
7. Briefly describe the relationship between desk checking, structured walk-throughs, and design and code inspections.
8. What two factors should have a major impact on the way programming teams are organized?
9. When is information systems asset secure?
10. In the context of information systems assets, briefly explain the difference between physical security and logical security.
11. Briefly describe the four major tasks that must be undertaken during the exposures analysis phase of a security program.
12. Briefly discuss the responsibilities of security administrators with respect to maintenance of the supply of energy to the information systems function.
13. What are the controls of last resort? Briefly explain the nature of each.
14. What considerations affect the choice of a backup site?
15. What is the primary role of quality assurance management as it operates within the information systems function?
16. Give *three* reasons why the QA function has emerged in organizations.
17. Why is it best to strive for minimal specification of standards when preparing information systems standards?

18. Why should QA personnel be notified routinely of errors or irregularities that occur in information systems?
19. Why do QA personnel need to monitor national and international information systems standards?
20. Why is it important to include quality metrics in an information systems quality assurance project plan?

REFERENCE BOOKS

1. Weber R; Information Systems Control and Audit (Person Education)
2. Dube: Information systems for Auditing (TMH)
3. Auditing Information Systems, 2nd Edition. Jack J. Champlain (Wiley)

UNIT – III

1. Input Controls

Structure

Objectives

After going through this lesson, you should be able to:

- understand the data input methods
- understand Data screen design and controls
- understand how to Check digits impliment
- discuss about validation of data input
- Discuss about instruction input and validation
- understand how to Audit trail controls.

1.1 Introduction

Components in the input subsystem are responsible for bringing both data and instructions into an application system. Both types of input must be validated Moreover, any errors detected must be controlled so input resubmission is accurate, complete, unique, and timely.

This lesson examines controls over the input subsystem. From auditor's viewpoint, input controls are critical tor three reasons. First, in many information systems the largest number of controls exists in the input subsystem. Consequently, auditors will often spend substantial time assessing the reliability of input controls. Second, input subsystem activities sometimes involve large amounts of routine, monotonous human intervention. Thus, they are error-prone. Third, the input subsystem is often the target of fraud. Many irregularities that have been discovered involve addition, deletion, or alteration of input transactions.

1.2 Data input methods

Recall that a typical way auditors evaluate controls in an application system is to trace instances of material transaction types through the system. If they are to undertake this task, however, they must first understand how the application system obtains its data input.

Figure 1-1 provides an overview of the diverse ways in which data input can be entered into an application system. In all cases, auditors are interested in recording the *state* of some object or thing or an *event* that has occurred to some object or thing. For example, auditors might want to record a person's pay rate (a state) or an order placed by a customer (an event).

In many cases, data about the state or event will be input directly to a system (the lower branch of Figure 1-1). For example, a customer might approach a sales counter and place an order, and an order-entry clerk might then key data about the order into a terminal. Beside keyboard-based input devices, many other types now exist that allow direct-entry input of data for example, touch screens, mice, joysticks, trackballs, voice, video, and sound. Some types of devices will even track eye movements and gestures that a person makes with their hands.

Data about a state or event might also be recorded on some medium before it is entered into a system (the upper branch of Figure 1-1). For example, a salesperson might transcribe data about a sale onto a source document: data about a person's account number might be recorded on the magnetic strip on the back of a plastic credit card: a universal product code (UPC) might be printed on the packaging material for some product. In some cases, the medium then might be read directly by an input device for example, an optical character reader or imaging device. In other cases, data recorded on the medium might be keyed into a terminal for example; a data-entry operator keys data off source documents into a microcomputer. When this input method is used, *data preparation activities* are often undertaken prior to entering the data into the system. For example, source documents might he scanned for authenticity, accuracy, and completeness prior to their being keyed at a terminal. Similarly, annotations and notes might need to be erased and paper clips and staples might need to be removed before documents are read by an imaging device. The documents might then be sorted and assembled into batches.

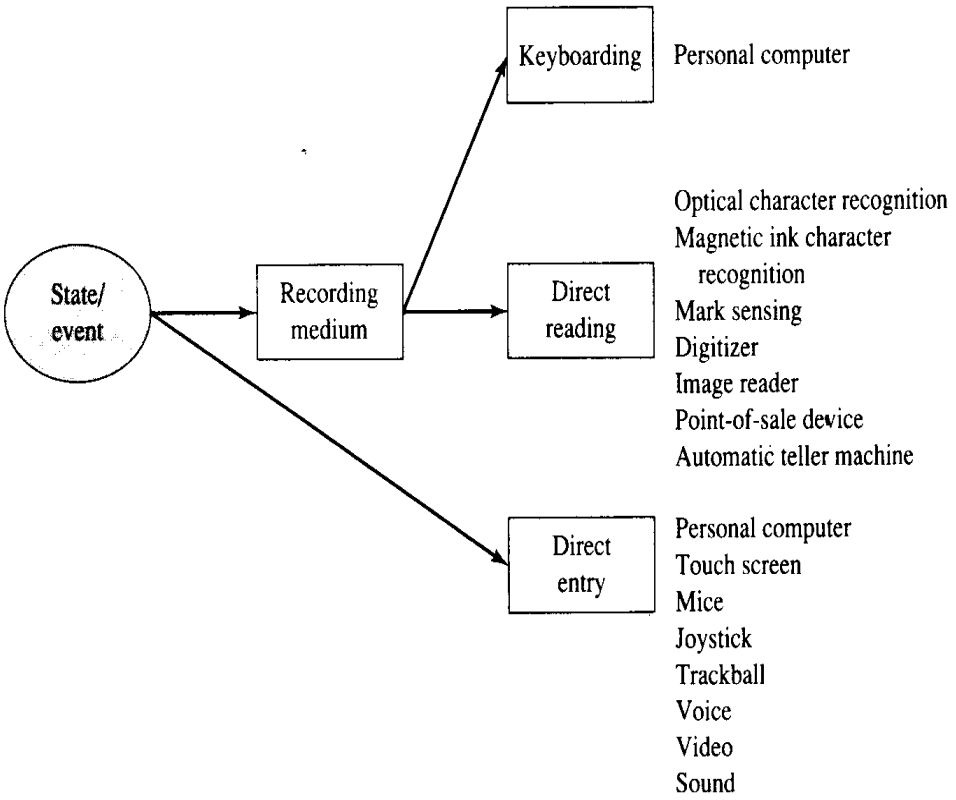


FIGURE 1-1 Input methods

When data is recorded on some medium prior to input to an application system, either prerecording or postrecording of the state or event can occur. Under *prerecording,* the state or event is recorded on the medium before the existence of the state is detected or the event has occurred. For example, an organization might record on a plastic credit card that one of its customers is always entitled to a discount when the customer purchases goods (prerecording of a state), or the characteristics of a clothing item and its price might be recorded on a tag so that these data can be read using an optical wand at the time of sale (prerecording of an event). Under *postrecording,* the state or event is recorded on the medium after the existence of the state has been detected or the event has occurred. For example, a person's marital status might he recorded on a source document during an interview (postrecording of a state), or a sale might be recorded on a source document after it has occurred (postrecording of an event).

Combinations of data input methods can also be used. For example, when customers use an automatic teller machine (ATM), the machine reads data from the plastic card (e.g., an account number) that customers insert into the machine (prerecorded, medium-based data entry). Customers also use the machine's keypad to enter the amount they wish to withdraw from their accounts (direct entry).

TABLE 1-1 Control Advantages of Using Point-of-Sale Terminals

---

1. Optical scanning of a premarked code improves pricing accuracy.
2. Customers can verify accuracy and completeness of a sale because they can be provided with a detailed receipt.
3. Improved control over tender because the terminal controls the cash drawer, automatically dispenses change and stamps, and handles any types of tender cash checks, coupons, food stamps.
4. Automatic check authorization or credit-card authorization. Customers can also enter PINs to authorize funds transfer from their accounts to the vendor's account.
5. Maintenance of independent records of transactions undertaken via journal tapes.
6. Better inventory control through more timely information on item sales.

TABLE 1-2 Control Advantages of Using Automatic Teller Machine

---

1. Physical security over cash. Many have antitheft features like alarms, camera surveillance, movement indicators, and heat detectors.
2. Maintenance of independent records of transactions undertaken via journal tapes and control counters.
3. Cryptographic facilities to preserve the privacy of data entered.
4. Software to guide customers through the input process, thereby minimizing errors or omissions.

By understanding the types of data input methods used within an application system, auditors can develop expectations about the probable control strengths and weaknesses in the input subsystem. Consider, for example, the following three aspects of input methods and how they are likely to affect auditors' assessment of control strengths and weaknesses:

1. As the amount of human intervention in the data input method increases, the likelihood of errors or irregularities occurring increases. For example, if sales data is first recorded onto a source document and then keyed into an application system, there is a greater chance of errors occurring than if the data is keyed immediately into a microcomputer or scanned from a tag. In the former case, errors can occur during transcription as well as during keying, whereas in the latter two cases, errors can occur only during keying or scanning. In general, relative to direct-entry input or prerecorded, medium-based input methods, post-recorded, medium-

based input methods are more prone to errors and irregularities because they are more labor intensive.

2. As the time interval between detecting the existence of a state or event and input of the state or event to an application system increases, the likelihood of errors or irregularities occurring increases. Data about the state or event could be forgotten; or the person who enters the data might not be the person who observed the state or event, and thus he or she might be less able to respond to queries about the validity of the data. As the time interval lengthens, more opportunities also arise for data about the state or event to be -altered improperly.

3. Use of certain types of input devices facilitates control within the input subsystem because they possess characteristics that militate against errors or irregularities. For example, Tables 1-1 and 1-2 list some of the major control advantages that arise when point-of-sale terminals and automatic teller machines are used as input devices.

1.3 Source document design

Some data-input methods use source documents to record data that will be entered into a computer system. Source documents are often used when there will be a delay between capturing the data about a state or event and input of that data into a computer system. For example, sales data might be captured remotely and mailed to the head office of an organization for input to an order-entry system. Source documents are also used as turnaround documents. For example, a computer-prepared remittance advice could be sent to customers with their invoices. Customers are asked to return the remittance advice along with their payment. The remittance advice is then read optically into an accounts receivable system.

From a control viewpoint, a well-designed source document achieves several purposes:

1. It reduces the likelihood of data recording errors;
2. It increases the speed with which data can be recorded;
3. It controls the work flow;
4. It facilitates data entry into a computer system;
5. For pattern recognition devices, it increases the speed and accuracy with which data can be read; and
6. It facilitates subsequent reference checking.

Auditors, therefore, must understand the fundamentals of good source document design. Source document design begins after carrying out source document analysis. Source document analysis determines what data will be captured, how the data will be captured, who will capture the data, how the data will be prepared and entered into a computer system, and how the document will be handled, stored, and filed.

After these requirements have been determined, two decisions can be made. First, the characteristics of the paper medium to be used for the

source document can be chosen. This decision involves selecting the length and width of the paper, its grade and weight, and whether single-part or multipart paper will be used. If the wrong length and width is chosen, for example, the source document might be difficult to handle or to read via some type of scanning device. If the wrong grade and weight are chosen, the paper could smudge or tear under adverse conditions. If the wrong decision is made on whether to use single- or multipart paper, multiple recordings of the same source data could have to be made, or alternatively the multipart paper chosen to avoid multiple recordings could tear because it is too thin.

The second decision relates to the layout and style that will be used for the source document. The choice of layout and style has an important impact on the number of input errors that will be made using the source document. Some important design guidelines follow:

1. *Preprint wherever possible.* Preprint all constant information on a source document. If only a limited number of responses to a question is appropriate, preprint the responses and have the user tick or circle the correct responses or delete those that are inappropriate.

2. *Provide titles, headings, notes, and instructions.* A title clearly identifies the purpose of the source document. Headings break up the document into logical sections. Notes and instructions assist the user to complete the document. Where codes are used preprint their meaning on the form so the user does not have to rely on memory or waste time looking up reference manuals.

3. *Use techniques for emphasis and to highlight differences.* Different type fonts such as italics and boldface give emphasis to different parts of the source document. Heavy thick lines or hatching highlight important fields or sections of the source document. Different colors facilitate distribution of different copies of the source document. Background colors emphasize special sections of the source document for example, those for office use only.

4. *Arrange fields for ease of use.* Design the source document to be completed in a natural sequence from left to right, top to bottom. Group related items together. The sequence of fields should follow the work flow: The most used fields on the left of the document; those usually used in the center; and those seldom used on the right.

5. *Use the "caption above fill-in area" approach for captions and data fields.* Figure 1-2a shows three approaches to designing the layout of captions and fields on a source document: caption preceding the fill-in area, caption within the fill-in area, and caption above the fill-in area (floating box). Galitz argues that the "caption above fill-in area" or floating-box approach is the best. The caption should be centered above the fill-in area or left justified above the fill-in area if the fill-in area is long.

6. *When possible, provide multiple-choice answers to questions to avoid omissions.* Figure 1-2b shows how this technique can be used with the floating-box approach. Rather than ask users to remember all the business subjects they studied, provide a list they can check.

7. *Use tick marks or indicator values to identify field-size errors.* Figure 1-2c shows how these techniques highlight field overflow or underflow. Tick marks can be used when a field must contain a fixed number of characters. Indicator values can be used to show the maximum when a variable number of characters can be inserted in the field.

8. *Combine instructions with questions.* Figure *1-2d* shows how this technique overcomes possible confusion about the format for a date.

9. *Space items appropriately on forms.* Correct spacing of fields on forms is particularly important if responses are to be typewritten (e.g., the form is input to a laser printer to receive output from a word processing package).

10. *Design for ease of keying.* Have the order in which fields are keyed follow the order of field placement.

11. *Prenumber source documents.* Prenumber source documents so users can account for every document. If each document has a unique serial number, input transactions can be sorted by serial number and breaks in the sequence of numbers identified.

12. *Conform to organizational standards.* An organization should have a forms control section responsible for overall forms design standards; for example, numbering and color conventions, placement of the organization's logos, retention requirements, and ordering and stock keeping requirements. Ensure the source document design conforms to these standards.

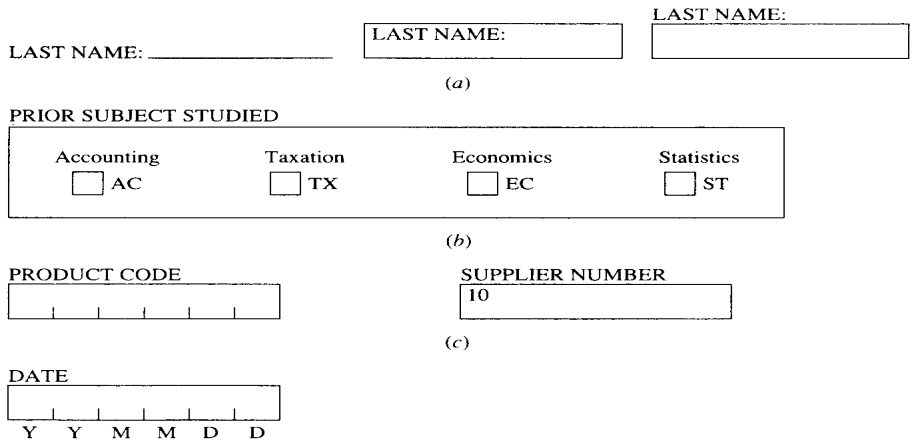


FIGURE 1-2(a) Caption preceding fill in area, caption within fill-in area, and caption above fill in area source document design approaches. (b) using multiple choice to prevent omission (c) using tick marks and indicator values to identify field size errors (d) combining instructions with questions

1.4 Data screen design

If data is keyed into a system via a terminal, high-quality screen design is important to minimizing input errors and to achieving an effective and efficient input subsystem. Auditors must be able to examine the data-entry screens used in an application system and to come to a judgment on the frequency with which input errors are likely to be made and the extent to which the screen design enhances or undermines effectiveness and efficiency. This judgment will affect the way they decide to conduct the remainder of the audit.

The following subsections provide a brief introduction to screen design issues and are based primarily on Galitz, Weinschenk and Yeo, Mullet and Sano, and Horton. Certain design principles apply to all types of data-entry screens. Others vary, however, depending on whether the screen is used for direct-entry input or input of data already captured on a source document. Interestingly, different authors often have conflicting recommendations on what constitutes good screen design. For example, some recommend that boxes be placed around data-entry fields, whereas others recommend use of the underscore character. Similarly, some recommend that field captions always be left-aligned, whereas other recommend that they be right-aligned if the size of the captions is markedly different. Auditors ultimately must make a judgment about the quality of the data-entry screen design confronted. Currently there are guidelines to assist auditors' judgment, but there are no fixed design rules.

1.4.1 Screen Organization

Screens should be designed so they are uncluttered and symmetrically balanced. The data elements should be organized into functional, semantic groups. Boxes can be used to highlight certain groupings of data elements.

As the number of rows, vertical alignment points, and data elements on a screen increases, the complexity of the screen increases. Using a measure of complexity proposed by Galitz, the screen in Figure 1-3a is more complex than the screen in Figure 1-3b. Thus, the former screen is likely to be more error prone than the latter. In addition, data entry using the former screen is likely to be slower.

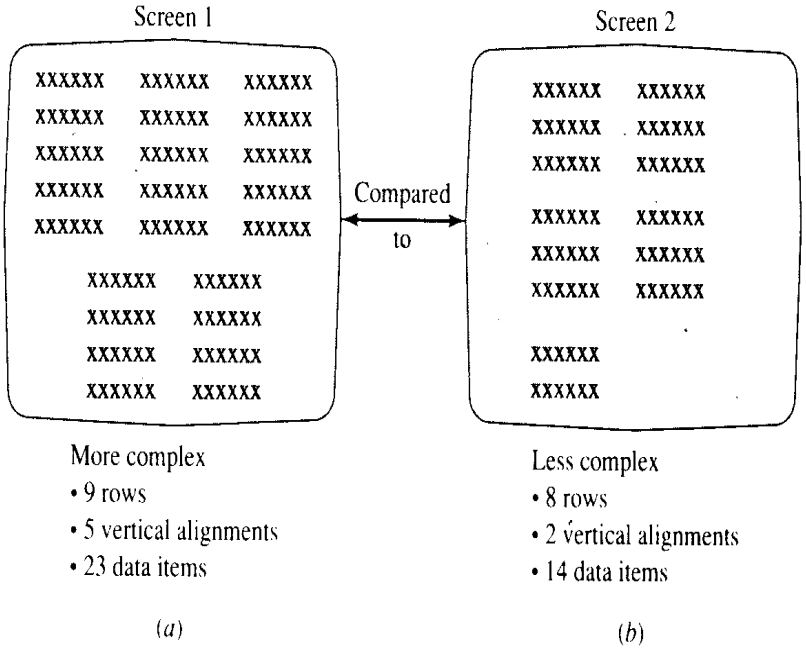


FIGURE 1-3 Evaluating screen complexity

All the information needed to perform a task must be on a screen, yet users still should be able to identify quickly the information they require. Where multiple screens must be used to capture a transaction, the screens should be broken at some logical point. Symmetry can be achieved by grouping like elements together, balancing the number of elements on both sides of the screen, ensuring elements are aligned, and using blank space and line delimiters strategically.

If a screen is used for direct-entry input of data, the layout of elements on the screen must mirror the way in which data is obtained during the data-capture task. If the screen is used for source-document data entry, however, the screen must be an image of the source document on which the data is first captured and transcribed. In the former case, the screen guides users through the data-capture process. In the latter case, users should be able to keep their eyes on the source document during the keying process and be required to view the screen only when they encounter some problem.

An important objective in screen design is consistency. Users develop facility with a particular design. Consequently, whenever possible this design should be used repeatedly across applications. For example, certain parts of a screen should always be used to display instructions for completing the screen, error messages, instructions for screen disposition, and status messages.

1.4.2 Caption Design

Captions indicate the nature of the data to be entered in a field on a screen. Design considerations include structure, size, type font, display intensity,

format, alignment, justification, and spacing. Again, the primary factor affecting the design of captions is whether the screen is used for direct-entry input of data or input of data already captured on a source document.

Captions must be' fully spelled out if a screen is used for direct-entry data capture. Because the screen guides the user during the data-capture process, the meaning of the captions must be unambiguous. If data entry is based on a source document, however, captions can be abbreviated because users can refer to the source document to obtain the full meaning of a caption.

Captions must be distinguished clearly from their associated data-entry field. For example, uppercase type font might be used for all captions, and lowercase type font might be used for the data entered by a keyboard operator. To further differentiate captions and data-entry fields, different display intensities can be used. Because captions are the primary focus during data entry, they should have higher display intensity than the data entered by users. Alternatively, captions and data-entry fields can be displayed using different colors.

Captions should always precede their associated data-entry field on the same line as the data-entry field. One exception to this guideline is where multiple data-entry fields relate to the same caption. In this case, the data entry fields should be stacked under the caption. For example:

SALESPERSON: _____ PREVIOUS OCCUPATION
_____
_____
_____

Where the caption and data-entry field appear on the same line the caption should be followed immediately by a colon. At least one space also should exist between the colon and the data-entry field. Where the data-entry fields are stacked, however, a colon is not needed before each data-entry field (providing underscores indicate the field position and size).

If direct-entry input of data is used, captions should be aligned vertically in columns. Within a column, Galitz argues that either *(1)* both captions and data-entry fields should be left justified or *(2)* captions should be right justified or data-entry fields should be left justified. For example:

NAME: _____ NAME: _____
AGE: _____ AGE: _____
POSITION: _____ POSITION: _____

If the screen is used for entry of data already captured on a source document, however, alignment and justification are dictated by the source document. The screen design should be an image of the source document.

Both horizontal and vertical spacing around captions are important to attaining an uncluttered screen. For horizontal spacing, direct-entry data capture screens should have a minimum of five spaces between the longest data entry field in a column and the leftmost caption in an adjacent column. Source document screens should have a minimum of three spaces between a data-entry field and the following caption. For vertical spacing,

direct-entry data capture screens should have a blank line every fifth row; that is, captions and the associated data-entry fields should be clustered in groups of five. Source-document screens, on the other hand, should mirror the vertical spacing found on the source document.

1.4.3 Data-Entry Field Design

Data-entry fields should immediately follow their associated caption either on the same line or, in the case of a repeating field, on several lines immediately below the caption. The size of a field should be indicated by using an underscore character or some other character. As each new character is entered into the field, the existing character is replaced. Alternatively, the size of a field can be indicated by using a lined box filled in with a contrasting color or background reports, however, that empirical studies have shown the underscore character is the best means of indicating field size.

Where direct-entry data capture screens are used, completion aids can be used to reduce keying errors. For example, if a date must be entered, either the caption or the field-size characters can be used to indicate the date format:

DATE(YYMMDD):_____ DATE: YYMMDD

On the other hand, where source-document screens are used, completion aids are not needed because the keyboard operator can refer to the source document for completion instructions.

Radio buttons, check boxes, list boxes, and spin boxes are now frequently used for direct-entry data capture. Radio buttons and check boxes should be used only if one or a small number of options exists. With long lists of options, list boxes can be used. Spin boxes can be used to cycle through a limited number of options.

1.4.4 Tabbing and Skipping

Galitz argues that automatic skipping to a new field should be avoided in data-entry screen design for two reasons. First, with an automatic skip feature, keyboard operators might make a field-size error that remains un-detected because the cursor simply skips to a new field. The additional character inadvertently added to the field will affect the character positions of all other fields, thereby causing them to be in error. Second, in many applications, data-entry fields often are not filled anyway. Thus, keyboard operators must still tab to the next field. Rather than having keyboard operators decide whether tabbing is needed, it is simpler to require them always to tab to the next field. Although the tab requires an additional keystroke, the operator's keying rhythm is maintained.

1.5 Data code controls

Data codes have two purposes. First, they uniquely identify an entity or identify an entity as a member of a group or set. Textual or narrative description does not always uniquely identify an entity: for example, two people could have the same name. Second, for identification purposes,

codes often are more compact than textual or narrative description because they require fewer characters to carry a given amount of information.

Poorly designed codes affect the input process in two ways: They are error prone, and they cause recording and keying processes to be inefficient. Auditors, therefore, must evaluate the quality of the coding systems used in application systems to determine their likely impact on data integrity, effectiveness, and efficiency objectives.

1.5.1 Data Coding Errors

There are five types of data coding errors:

1. *Addition.* An extra character is added to the code, e.g., 87942 coded as 879142.
2. *Truncation.* A character is omitted from the code, e.g., 87942 coded as 8792.
3. *Transcription.* A wrong character is recorded, e.g., 87942 coded as 81942.
4. *Transposition.* Adjacent characters of the code are reversed, e.g., 87942 coded as 78942.
5. *Double transposition.* Characters separated by one or more characters are reversed, e.g., 87942 coded as 84972.

Five factors affect the frequency with which these coding errors are made:

1. *Length of the code.* Longer codes are more error prone. The notion that length could be important derives from the work of Miller (1956). He argues that humans effectively can hold only about five to nine (average seven) "chunks" of information in short-term memory. In this light, long codes should be broken up into chunks by using, for example, hyphens, slashes, or spaces to reduce coding errors.

2. *Alphabetic/numeric mix.* If alphabetic and numeric characters are to be mixed in a code, the error rate is lower if the alphabetic are grouped together and the numeric's are grouped together. Thus, a code such as ABN653 is less error prone than the code A6BS3N. The latter code is also harder to key because it breaks the keying rhythm by interchanging alphabetic and numeric's.

3. *Choice of characters.* If possible, the characters B, I, 0, S, V, and Z should be avoided in codes because they are frequently confused with the characters 8,1, 0, 5, U, and 2. Also, the letters Y and N often are illegible when they must be handwritten.

4. *Mixing uppercase/lowercase fonts.* Having to use the shift key during keying of a code breaks the keying rhythm and increases the likelihood of error. If possible, only an uppercase or a lowercase font should be used for a code. Special characters, such as @, *, and # also cause problems because the shift key must be used.

5. *Predictability of character sequence.* Some character sequences are more predictable than others and, as such, are less error prone. For

example, the character sequence THE is more predictable than the character sequence ZXQ.

1.5.2 Types of Coding Systems

Specific codes are chosen within the context of a coding system. Ideally, a coding system achieves five objectives:

Objective	*Explanation*
Flexibility	A code should allow easy addition of new items or categories.
Meaningfulness	Where possible, a code should indicate the values of the attributes of the entity.
Compactness	A code should convey maximum information conveyed with the minimum number of characters.
Convenience	A code should be easy to encode, decode, and key.
Evolvability	Where possible, a code can be adapted to changing user requirements.

Unfortunately, it is impossible to achieve all these objectives simultaneously. The following subsections briefly examine four types of coding systems that auditors will encounter frequently and the extent to which they achieve these objectives.

1.5.3 Serial Codes

Serial coding systems assign consecutive numbers (or alphabetic) to an entity, irrespective of the attributes of the entity. Thus, a serial code uniquely identifies an entity. The code indicates nothing further about the entity, however, such as the category of items to which it belongs.

The major advantages of a serial code are the ease with which a new item can be added and conciseness. The low mnemonic value of serial codes can also be an advantage in some circumstances. For example, consider a database environment in which extensive sharing of data occurs and the number and types of users and their needs are in a state of flux. Different users might wish to view data differently. A code that presumes one view of data might be inappropriate for certain users. Thus, a serial coding system might contribute better to the evolution of the system.

The code presents problems when the file of items is volatile that is, significant numbers of additions and deletions occur. Deleted items must have their codes reassigned to new items; otherwise, significant gaps in the sequence occur, and the code is no longer concise. Users can become confused if the codes are constantly reassigned to new entities. Because the codes have no mnemonic value, they are also difficult to remember.

1.5.4 Block Sequence Codes

Block sequence codes assign blocks of numbers to particular categories of an entity. The primary attribute on which entities are to be categorized must be chosen, and blocks of numbers must be assigned for each value of the attribute. For example, if account numbers are assigned to customers on the basis of the discount allowed each customer, a block sequence code would look like this:

101 R. Allen 102 J. Smith 103 M. Clarke	3% discount allowed
201 S. Elders 202 M. Ball	3½% discount allowed
301 K. Kline 302 G. Brown 303 F. Water	4% discount allowed

Block sequence codes have the advantage of giving some mnemonic value to the code. Nevertheless, choosing the size of the block needed can be difficult (and the remedy if overflow occurs). If the block sizes are too large, characters are wasted, and the code is no longer concise. Long codes are more difficult to remember.

1.5.5 Hierarchical Codes

Hierarchical codes require selection of the set of attributes of the entity to be coded and their ordering by importance. The value of the code is a combination of the values of the codes for each attribute of the entity. For example, the-following hierarchical code for an account has three components (expenditure within departments within divisions):

C65	423	3956
Division number	Department number	Type of expenditure

Hierarchical codes are more meaningful than serial or block sequence codes because they describe more attributes of the entities to which they apply. Thus, they should be easier to recall. Nevertheless, they are not always concise. As a result, their length undermines recall accuracy. Sometimes they also mix alphabetics and numerics, which again undermines recall accuracy.

Hierarchical codes sometimes present problems when change occurs. In the example given previously, consider the implications of a change to the organizational structure for example, department 423 might be assigned to

a different division C25. The codes for many items might have to be altered, and in some cases extensive re-sequencing of master files might have to occur.

1.5.6 Association Codes

With an association code, the attributes of the entity to be coded are selected, and unique codes are then assigned to each attribute value. The codes can be numeric, alphabetic, or alphanumeric. The code for the entity is simply the concatenation of the different codes assigned to the attributes of the entity. Unlike an hierarchical code, however, the order in which the codes for the attributes occur in the overall code does not necessarily imply some type of hierarchical relationship.

The following is an example of an association code assigned to a shirt:

SHM32DRCOT
where SH = shirt
M = male
32 = 32 centimeters, the neck size
DR = dress shirt
COT = cotton fabric

Association codes have high mnemonic value. They can be error-prone, however, if they are not concise or they contain too much of a mixture of alphabetic and numeric characters. Error-prone characters, such as I, 0, and S, also might have to be used if the code is to have mnemonic value.

1.6 Check digits

In some cases, errors made in transcribing and keying data can have serious consequences. For example, keying the wrong stock number can result in a large quantity of the wrong inventory being dispatched to a customer. Aside from the costs of retrieving the inventory and replacing it with the correct inventory, customer goodwill can be lost. One control used to guard against these types of errors is a check digit. Check digits are now used as a means of detecting errors in many applications for example, airline ticketing, credit card processing, bank account processing, blood bank item processing, and driver's license processing.

1.6.1 Nature of Check Digits

A *check digit* is a redundant digit(s) added to a code that enables the accuracy of other characters in the code to be checked. The check digit can act as a prefix or suffix character, or it can be placed somewhere in the middle of the code. When the code is entered, a program recalculates the check digit to determine whether the entered check digit and the calculated check digit are the same. If they are the same, the code is most likely to be correct. If they differ, the code is most likely to be in error.

1.6.2 Calculating Check Digits

There are many ways of calculating check digits. A simple way is to add up the digits in a number and assign the result as a suffix character. For

example, if the code is 2148, the check digit is 2 + 1 -I- 4 + 8 = 15. Dropping the tens digit, the check digit will be 5 and the code 21485. This check digit does not detect a very common kind of coding error, however namely, a transposition error. The incorrect code 2814 still produces the correct check digit.

To overcome this problem a different method of calculating a check digit can be used. The approach assigns different weights to different digits in the code. Given, again, the code 2148, the steps are as follows:

1. Multiply each digit by a weight. Assume the weight used will be 5-4-3-2; that is, 2 for the units digit, 3 for the tens digit, 4 for the hundreds digit, and 5 the thousands digit, viz:

8X2 =16
4X3 =12
1x4=4
2x5 =10

2. Sum the products = *42.*

3. Divide by a modulus. In this case, assume we choose the modulus 11:

42 -.-.-= 3 with remainder 9

4. Subtract the remainder from the modulus and the result constitutes the check digit.

11-9=2

5. Add the check digit to the code as a suffix. The result is 21482.

The check digit can be recalculated upon keying to detect a coding or key-ing error, or upon reading the data into the computer. The recalculation for this code proceeds as follows:

1. Multiply each digit by its corresponding weight. The check digit takes a weight of 1.

2X1=2
8X2 =16
4X3 =12
1X4=4
2X5 =10

2. Sum the products = 44.

3. Divide by the modulus (44/11)=4

4. If the remainder is zero, there is a high probability the code is correct.

If the code contains alphabetics or a special character (such as a hyphen), a check digit can still be calculated. Each alphabetic or special character must be assigned a number according to some rule.

1.6.3 When to Use Check Digits

Overheads arise from using check digits because a redundant character must be carried at least partially through the system. Extra computation is also needed to calculate and validate the check digit. Therefore, use of check digits should be limited to critical fields.

Manual calculation of or checking of check digits should be avoided. The process is time-consuming and error-prone. For new codes, the check digit should be precalculated automatically and assigned as part of the code.

Validation of check digits should be undertaken only by a machine for example, during keying or by an input program. To save storage space, the check digit can be dropped after it has been read by an input program and recalculated upon output. The trade-off here is storage space versus processing time.

1.7 Batch controls

Some of the simplest and most effective controls over data capture and entry activities are batch controls. Batching is the process of grouping together transactions that bear some type of relationship to each other. Various controls then can be exercised over the batch to prevent or detect errors or irregularities.

1.7.1 Types of Batches

There are two types of batches: physical batches and logical batches. *Physical batches* are groups of transactions that constitute a physical unit. For example, source documents might be obtained via the mail, assembled into batches, spiked and tied together, and then given to a data-entry clerk to be entered into an application system at a terminal. Similarly, documents that are to he input to an electronic imaging system might be assembled into batches before they are scanned or filmed.

Logical hatches are groups of transactions bound together on some logical basis, rather than being physically contiguous. For example, different clerks might use the same terminal to enter transactions into an application system. Clerks keep control totals of the transactions that they have entered. The input program logically groups transactions entered on the basis of the clerk's identification number. After some period has elapsed, it prepares control totals for reconciliation with the clerk's control totals.

1.7.2 Means of Batch Control

Two documents are needed to help exercise control over physical batches: a batch cover sheet and a batch control register. A batch cover sheet contains the following types of information:

1. A unique batch number:
2. Control totals for the batch:
3. Data common to the various transactions in the batch, e.g. transaction type;

4. Date when the batch was prepared;

5. Information on errors detected in the batch: and

6. Space for signatures of personnel who have handled the batch in some way, e.g. the person who prepared the batch and the person who keyed the hatch.

A batch control register records the transit of physical batches between various locations within an organization. Each person responsible for handling batches has a batch register. The register is signed each time a batch is received or dispatched. The person who brings the batch or takes it away countersigns the register. In some cases the person taking over responsibility for the batch also checks its contents. If a dispute arises over the location of a batch, the batch control registers can be consulted.

1.8 Validation of data input

Data submitted as input to an application system should be validated as soon as possible after it has been captured and as close as possible to its source. Errors then can be corrected by persons who are likely to have most knowledge about them and while the circumstances surrounding the data are still fresh in their minds. Sometimes this objective cannot be achieved. For example, the data in error might have been received from a remote organization via an electronic data interchange system. Some time could elapse before the receiving organization resolves the error with the sending organization.

Any errors identified that are not corrected immediately should be written to an error file (Figure 1-4). Otherwise, users might forget to correct the errors. The input subsystem should use the error file to remind users when they have not corrected errors on a timely basis. Reminder messages can be displayed on screens or printed on hard-copy reports.

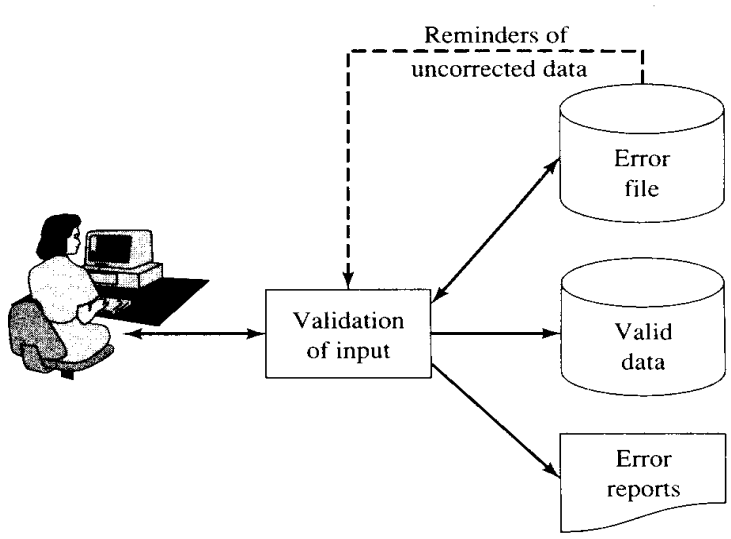


FIGURE 1-4 Use of an error file for data validation

1.8.1 Types of Data Input Validation Checks

To some extent the types of data input validation checks undertaken depend on the nature of the data input method used. For example, if documents are scanned via an imaging device, a quality-control person should be responsible for examining document images, comparing them against the original documents, and rejecting the image or undertaking cleanup work if its quality is unacceptable.

To illustrate the nature of data input validation checks, however, consider four types that can be undertaken when input data that is keyed in at a terminal: *(1)* field checks, *(2)* record checks, *(3)* batch checks, and *(4)* file checks. Each is discussed briefly in the following subsections.

Field checks

With a field check, the validation tests applied to the field do not depend on other fields within the input record or within other input records. For example, we can check whether a field that is supposed to contain numeric data only does, in fact, contains only numeric characters.

Record Checks

With a record check, the validation tests applied to a field depend on the field's logical interrelationships with other fields in a record. For example, auditors can check whether the range of salary values in one field is reasonable given the value of another field that indicates a person's seniority.

Batch Checks

With a batch check, the validation tests examine whether the characteristics of a batch of records entered are congruent with the stated characteristics of the batch. For example, auditors can check whether the total of all financial .fields in the batch of records equals the grand total given for the batch.

File Checks

With a file check, the validation tests examine whether the characteristics of a file used during data entry are congruent with the stated characteristics of the file. For example, if auditors validate some of the characteristics of data that is keyed into an application system against a master file, they can check whether they are using the latest version of the master file.

1.8.2 Reporting Data Input Errors

Errors must be reported by the input validation program in a way that facilitates fast and accurate correction of the errors. Errors can be signaled via a buzzer or bell. The cursor also can be made to flash to show the data item in error. An error message should then be displayed to indicate the nature of the error and possible corrective actions that might be undertaken. Error messages must be designed carefully to be

1. *Clear and concise.* Messages should use short, meaningful, and familiar words, avoid the passive voice, avoid contractions and abbreviations, and issue instructions in the sequence to be followed.

2. *Courteous and neutral.* Messages should avoid familiarity, be polite and instructive, avoid humor or condemnation, and assist the user to solve the problem even if repeated errors are made.

1.9 Instruction input

Ensuring the quality of instruction input to an application system is a more difficult objective to achieve than ensuring the quality of data input. Data input tends to follow standardized patterns. The errors or irregularities that are likely to occur usually can be anticipated. During instruction input, however, users often attempt to communicate complex actions that they want the system to undertake. On the one hand, the input subsystem must provide considerable flexibility so users can accomplish their processing objectives. On the other hand, it must exercise careful control over the actions they undertake. The approaches used to communicate instructions to an application system tend to trade off flexibility with control.

1.9.1 Menu-Driven Languages

The simplest way for users to provide instructions to an application system is via a menu. The system presents users with a list of options. Users then choose an option in some way for example, by typing a number or letter to indicate their choice, positioning the cursor on the selection and pressing the return key, pressing or releasing a button on a mouse, using a light pen, or touching the screen with their ringer,

Different types of menus can be used. Menu bars contain items that always appear on a screen. They provide major guidance for users when interacting with a screen. Pull-down menus contain items that are used less frequently. They disappear, for example, when users release a mouse button. Pull-down menus often lead into cascading menus where highlighting one menu item leads to a list of subsidiary menu items then being displayed. Popup menus are used to provide users with a limited set of actions specific to a certain place where they are located on a screen or a particular action they are taking.

The following guidelines should reduce the number of errors that are likely to occur using menu input.

1. Menu items should be grouped logically so they are meaningful and memorable.

2. Menus with greater breadth and less depth are usually faster to use and less error prone than menus with greater depth and less breadth.

3. Menu items should follow any natural order that exists. If no natural order exists, short menus are often best ordered by frequency of occurrence and long menus by alphabetical order.

4. Menu items that appear in more than one menu should retain the same position within the different menus.

5. Menu items should be fully spelled, clear, concise, verbs or nouns or verb-noun pairs.

6. The basis for selecting a menu item should be clear for example, numbers (starting with one, not zero), a mnemonic abbreviation, or a radio button.

7. Where other output is displayed on a screen, the menu should be clearly differentiated. Alternatively, pull-down or pop-up menus can be used to hide some or the entire menu.

1.9.2 Question-Answer Dialogs

Question-answer dialogs are used primarily to obtain data input. The application system asks a question about the value of some data item, and the user responds. Nevertheless, question-answer dialogs also can be used to obtain instruction input in conjunction with data input.

If the answers to be provided in a question-answer dialog are not clear, users could make errors when they provide instruction (or data) input. A well-designed question-answer dialog makes clear the set of answers that are valid. In those cases in which the required answers are not obvious, a help facility can be used to assist inexperienced users.

1.9.2 Command Languages

Command languages require users to specify commands to invoke some process and a set of arguments that specify precisely how the process should be executed. For example, SQL is a database interrogation language that uses a command-language format. To print the customer numbers of those customers who had more than ten transactions over $200, the following SQL command sequence might be specified:

```
SELECT CUSTNO
FROM TRANS
WHERE AMOUNT > '200'
GROUP BY CUSTNO
HAVING COUNT (*) > '10';
```

In this example, "SELECT" is a command, and "CUSTNO" is an argument.

Two major decisions must be made in the design of command languages: first, whether to use a large number of commands with a small number of arguments or a small number of commands with a large number of arguments; and second, whether to use keywords or position to specify the arguments. These decisions affect how easy the language is to use and the number of errors users are likely to make.

In most situations, it appears better to use command languages with a small number of commands and a large number of arguments. Inevitably, users seem to employ only a small subset of the commands available in a command language perhaps because they have 'difficulty remembering all the commands. Thus, it seems better to make these commands powerful by providing an extensive list of arguments.

Whether arguments should be specified by keywords or position seems to depend on the user's expertise with the command language. With a little experience, presumably most users would prefer to type the following:

COPY MYFILE YOURFILE

instead of

COPY FROM = MYFILE TO = YOURFILE

Nevertheless, as argument lists become longer, remembering the position of each argument and whether it is mandatory or optional becomes more difficult. Keyword specification of arguments might then be preferred.

To facilitate recall of commands, command names should be meaningful. Moreover, commands that specify opposite actions should be congruent with one another in the sense of everyday usage of the commands. For example, when users wish to add characters to or remove them from a file, they are likely to prefer the commands INSERT/DELETE to the commands INSERT/OMIT.

To reduce typing effort, it should be possible to truncate commands. This strategy is easier to implement when only a small number of commands are used because truncations are likely to be unique. There are several ways to truncate commands for example, use the first and last letter of the command or delete vowels from the command. Whatever the truncation strategy used, it should be applied consistently across all commands.

Prompts and defaults reduce the number of errors made using a command language. For example, if users cannot remember the arguments associated with a command, they should be able to type a "?" or press some other key to obtain a prompt from the language on each argument required. Similarly, a command language reduces typing effort if it supplies the likely value of an argument as a default. For example, some spreadsheet command languages use the position of the current cell as the default value in many commands. The default can simply be overwritten if it is not the value required.

1.9.3 Forms-Based Languages

Forms-based languages require users to specify commands and data in the context of either some input or output form. if the output were some type of graph, users might employ a light pen to select a command that indicates they want the scales of the axes to be changed.

Forms-based languages can be successful if users solve problems in the context of input and output forms. In these cases the syntax of the language corresponds to the ways users think about the problem. As a result, input errors are reduced, and the language tends to be used

effectively and efficiently. When the functions to be performed do not map easily into the context of input and output forms, however, forms-based languages tend to be awkward and unwieldy to use.

1.9.4 Natural Languages

Natural language interfaces are still primarily the subject of substantial research and development efforts. Nevertheless, a few commercial products are now available. Auditors, therefore, might confront them increasingly in selected application domains and be required to evaluate their capabilities.

The ultimate goal of research on natural language interfaces is to enable relatively free-form natural language interaction to occur between users and an application system, perhaps via a speech production/recognition device. For certain types of applications, this objective might be laudable. Natural language might not be the best form of interface, however, for all types of applications. In particular, many of the sorts of applications that tend to concern auditors might not be suited to natural language interfaces.

Current natural language interfaces have several limitations:

1. They do not always cope well with the ambiguity and redundancy present in natural language. For example, the meaning of the sentence "Time flies." is different depending on whether "time" is the noun and "flies" is the verb or vice versa.

2. Substantial effort sometimes must be expended to establish the lexicon for the natural language interface. Users must define all possible words they could use, and this work must be redone each time a new application domain is to be accessed via natural language.

3. Even minor deviations outside the lexicon established for the application domain can cause problems. Users might be unaware of the precise boundaries of the domain and be inhibited in the commands they issue in case they traverse these boundaries.

4. If the database with which users interact is subject to frequent definitional changes, natural language interfaces can quickly become problematic. The lexicon must be able to evolve in light of definitional changes. Current lexicons do not always adapt well to changes in the database definition.

5. Users might be unaware of the ambiguity that can exist in natural language responses that the system gives to the commands they issue. For example, the query "How many stores in Tasmania had price overrides for sales of windsurfers?" might evoke a response of "none." If no stores in Tasmania are selling windsurfers, however, the response might be misleading.

6. It is still unclear whether the wordiness of natural language leads to ineffective and inefficient interaction with an application system. If users wish to express commands, therefore, in a formal, constrained, or

abbreviated way, a natural language interface should be able to accept this form of input.

7. Users still need some training when they employ natural language interfaces. Otherwise, they might ask queries that a natural language interface with even an extensive lexicon might not be able to interpret.

Until these technical problems have been overcome, auditors should be cautious in their evaluation of natural language interfaces. If it is critical that absolute precision be attained in the command and data input supplied to an application system and in the responses obtained from the system, other types of interfaces might be better.

1.10 Validation of instruction input

Like data input, instruction input entered into an application system also must be validated. Auditors might have little concern about the validity of instruction input when *(1)* the instructions are provided as part of a widely used application software package (e.g., the menu in an accounts receivable package) or *(2)* the instructions are interpreted via a high-level programming language (e.g., SQL commands in a database management system package). Where instruction input is designed and implemented specifically for a particular application system, however, auditors must evaluate the ways instructions are validated more carefully.

1.10.1 Types of Instruction Input Validation Checks

Three types of validation checks can be undertaken on input instructions: (7) lexical checks, *(2)* syntactic checks and (3) semantic checks. Each is discussed briefly in the following subsections.

Lexical Validation

During lexical validation, the system evaluates each "word" entered by a user. Three types of words can be encountered: *(1)* identifiers (labels, variables), *(2)* terminals (operators, reserved words), and *(3)* literals (numerical constants, strings). Because words are formed from characters, the system must establish rules whereby strings of characters are recognized as discrete words. Usually this recognition occurs via boundary characters and delimiters. For example, a space or an operator (*, /, +,—) might delimit a word.

To illustrate lexical analysis, assume the following SQL command is entered by a user:

```
SELECT name
FROM employee
WHERE salary > '15000'
```

The lexical analyzer in the system would read the command, character by character, and attempt to identify the words entered. For example, it would see that a space terminates the characters S, E, L, E, C, and T and that the character string "SELECT" is a reserved word within the language.

Similarly, a space terminates the variable "salary," the constant "15000" is delimited by the quotes symbol, and the variable "salary" and the constant " '15000' " are separated by an operator ">." If the lexical analyzer cannot recognize a valid word, it must print or display an error message so users can undertake corrective action.

Syntactic Validation

During syntactic validation, the system reads the string of words identified and validated by the lexical analyzer and attempts to determine the sequence of operations that the string of words is intended to invoke. For example, an instruction issued in an interactive command language might be the following:

INTEARN = (OLDBAL + DEPOSITS - WITHDRAWS)*INTEREST

The parentheses imply a particular sequence of operations, namely:

Add DEPOSITS to OLDBAL

Subtract WITHDRAWS from the result

Multiply the results by INTEREST

Store the result in INTEARN (interest earned)

Without the parentheses, the first action invoked might be to multiply WITH-DRAWS by INTEREST.

The syntax analyzer validates the syntax of an instruction by *parsing* the string of words entered to determine whether it conforms to a particular rule in the grammar of the language. Thus, the quality of syntactic validation depends on having a formal and complete description of the grammar on which the language is based and on making a good choice with respect to the parsing scheme chosen. Otherwise, errors in an instruction entered might not be identified or the error message displayed or printed might not be meaningful.

Semantic Validation

During semantic validation, the system completes its analysis of the meaning of the instruction entered. The boundary between syntactic validation and semantic validation is often obscure. During semantic validation, however, the language might check, for example, whether two variables that are to be multiplied together are numeric types and not alphabetic or alphanumeric types. Similarly, the system might prevent a comparison of two numeric values that would be meaningless for example, the salaries of employees with their weight.

The quality of semantic analysis depends on how well the constraints that relate to the data on which the instructions operate can be expressed. Database management systems that provide extensive data definition facilities, for example, allow high-quality semantic validation to be performed. The system can check that the operations to be undertaken on data items or the results produced conform to the constraints expressed in relation to the data items in the data definition.

Reporting Instruction Input Errors

The guidelines for reporting errors discussed earlier for data validation apply also to instruction validation. Error messages must communicate to users as completely and meaningfully as possible the nature of errors made during instruction input. Because the instructions that users enter could be variable and complex, substantial time can be lost if error messages do not allow users to pinpoint errors quickly. Multiple levels of error messages might be provided to cater for different levels of user expertise. Furthermore, if the system fails to identify an error, unbeknown to users, meaningless results can be produced.

1.11 Audit trail controls

The audit trail in the input subsystem maintains the chronology of events from the time data and instructions are captured and entered into an application system until the time they are deemed valid and passed onto other subsystems within the application system (e.g., the communications subsystem or the processing subsystem).

1.11.1 Accounting Audit Trail

In the case of data input, the accounting audit trail must record the origin of, contents of, and timing of the transaction entered into an application system. The types of data collected include the following:

1. The identity of the person (organization) who was the source of the data,
2. The identity of the person (organization) who entered the data into the system,
3. The time and date when the data was captured,
4. The identifier of the physical device used to enter the data into the system,
5. The account or record to be updated by the transaction,
6. The standing data to be updated by the transaction,
7. The details of the transaction, and
8. The number of the physical or logical batch to which the transaction belongs.

This data must be collected irrespective of whether the data was first captured on source documents, entered or read directly into the application system, or received from another organization via some type of inter organizational information system (say, an electronic data interchange system).

When input data is validated, a time and date stamp should be attached so the timing of data validation, error correction, and error resubmission subsequently can be determined. In some cases, a processing reference might be attached to the input data to indicate the program that performed the validation tests. In a distributed system, for example, input validation software could be replicated and executed at multiple sites. It might be

important to know which instance of the software performed the validation tests, particularly if doubts exist about consistency among replications.

If the input validation program identifies an error that cannot be corrected immediately, it must generate and attach a unique error number to the data in error. This error number must be associated with the data until it is corrected. It must be printed out or displayed on reports, entered on source documents used to correct the error, or keyed in at a terminal if the data is subsequently retrieved from the error file and corrected interactively. In this way the history of the erroneous data can be traced until the time of its correction.

In the case of instruction input, the audit trail might record the following types of data:

1. The identity of the originator of the instruction,
2. The time and date when the instruction was entered,
3. The identifier of the physical device used to enter the instruction,
4. The type of instruction entered and its arguments, and
5. The results produced in light of the instruction.

Like data, instructions entered in error can also be assigned a unique error number by the program that undertakes the validation. Unlike data, however, erroneous instructions often are not recorded on an error file that must be cleared. Instead, users simply reenter the instruction when they have determined the nature of the error they have made. If they do not reenter the instruction, the instruction is "lost." Any record of the instruction error typically is used for other purposes for example, analysis of the frequency with which different types of instruction errors are made.

2. Communication Controls

Structure

Objectives

After going through this unit, you should be able to:

- understand how to communication subsystem exposures
- understand how to Physical component controls
- understand how to Line error controls
- understand how to Flow controls and Link controls
- understand how to Topological controls
- understand how to Channel access controls and Controls over subversive threats
- understand how to Internetworking controls
- understand how to Audit trail controls

2.1 Introduction

The communication subsystem is responsible for transporting data among all the other subsystems within a system and for transporting data to or receiving data from another system. Its physical manifestation could be a cable (channel or bus) linking a disk drive with a central processor, or it could be a complex configuration of minicomputers, microcomputers, and communication lines) linking remote computers that must interact with one another.

Auditors are likely to spend increasing amounts of time evaluating control relating to the communication subsystem. The worldwide growth in communications traffic associated with computer systems has been dramatic, and it is likely to continue unabated for some time yet. Indeed, many organizations now could not survive in the absence of secure, effective, and efficient computer communication networks. We are also moving progressively toward information superhighways that will provide enormous capacity to transmit large volumes of voice, image, video, and data communications. Many organizations will have to make use of these superhighways if they are to survive competitively.

In this lesson we examine the types of controls that can be established within the communication subsystem to preserve asset safeguarding and data integrity. Although effectiveness and efficiency objectives are critical, we are able to consider them only in passing because they involve complex issues that warrant separate volumes. We focus, instead, on controls to reduce losses from failure in the subsystem components and deliberate attempts to subvert the authenticity and privacy of data traversing the subsystem components.

2.2 Communication subsystem exposures

Three major types of exposure arise in the communication subsystem: First, transmission impairments can cause differences between the data sent and the data received; second, data can be lost or corrupted through component failure; and third, a hostile party could seek to subvert data that is transmitted through the subsystem. The following subsections briefly examine each type of exposure.

2.2.1 Transmission Impairments

When data is transported across a transmission medium, three types of impairments can arise: attenuation, delay distortion, and noise. *Attenuation* is the weakening of a signal that occurs as it traverses a medium. It increases as the distance traveled by the signal increases. In the case of analog signals, *amplifiers* must be used after a signal has traveled a certain distance to boost the signal to a higher strength (amplitude). Otherwise, the receiver will not be able to detect and interpret the signal, or it will be corrupted by noise. In the case of digital signals, *repeaters* are used to boost the signal strength periodically as the signal traverses the medium. Attenuation can also cause *distortion* of analog signals. An analog signal is made up of a number of frequencies, and the amount of attenuation suffered varies across frequencies. Digital signals also suffer from attenuation distortion. They use a narrower range of frequencies, however, and thus the attenuation distortion that arises is less.

Delay distortion occurs when a signal is transmitted through bounded media (twisted-pair wire, coaxial cable, optical fiber). It does not occur when the signal is transmitted through air or space (free-space transmission). Different signal frequencies traverse bounded media with different velocities. Thus, signals are distorted because their different frequency components are subject to different delays. Delay distortion can have a marked effect on digital data because the signal energy in one bit position can spill over into another bit position.

Noise is the random electric signals that degrade performance in the transmission medium. There are four types of noise: white noise, inter modulation noise, crosstalk, and impulse noise. *White noise* (thermal noise) arises through the motion of electrons. It increases as a function of absolute temperature. *Intermodulation noise* arises when the output from a component in the communication subsystem is not a linear function of its

input. It can arise because of component malfunctioning. *Crosstalk* arises because signal paths become coupled. Bounded media are placed too close to each other, or the signal emitted by antennas used with unbounded media overlap. *Impulse noise* arises for a variety of reasons: atmospheric conditions (e.g., lightning), faulty switching gear, and poor contacts.

Noise increases as more data is transmitted over a medium. If the public telephone exchange network is used for data transmission, for example, line errors increase during peak periods because the increased traffic produces additional noise. Aside from the problems that arise with transmission errors caused by noise, wire tappers also can use noise to mask unauthorized activities.

2.2.2 Component Failure

The primary components in the communication subsystem are *(1)* transmission media for example, twisted-pair wire, optical fiber, and microwave; *(2)* hardware for example, ports, modems, amplifiers, repeaters, multiplexors, switches, concentrators; and *(3)* software for example, packet switching software, polling software, data compression software. Each of these components may fail. As a result, data in the communication subsystem may be lost, corrupted, or routed incorrectly through the network and perhaps displayed to a person who is unauthorized to view the data.

Hardware and software failure can occur for many reasons for example, failure in an integrated circuit, a disk crash, a power surge, insufficient temporary storage for a queue, or program bugs. The failure can be either temporary or permanent. For example, an intermittent failure in a modem could corrupt a bit pattern only in short bursts a temporary failure. Or an operating system could crash for some unknown reason, however, and the operator might be unable to restart it on a timely basis a permanent failure. The failure also can be either local or global. For example, the failure of a microcomputer terminal affects only the users of the terminal a local failure. Failure in a concentrator, however, affects all users connected to the concentrator a global failure.

2.2.3 Subversive Threats

In a subversive attack on the communication subsystem, an intruder attempts to violate the integrity of some component in the subsystem. For example, invasive or inductive taps can be installed on telephone lines using, say, a data scope. An *invasive tap* enables the intruder either to read or to modify the data being transmitted over the line. An *inductive tap* monitors electromagnetic transmissions from the line and allows the data to be read only. Similarly, satellite signals propagated in broadcast mode can be read by a ground receiver over a wide geographic area. Modifying satellite transmissions, on the other hand, is more difficult.

Subversive attacks can be either passive or active (Figure 2-1). In a *passive* attack, intruders attempt to learn some characteristic of the data

being transmitted. They might be able to read the contents of the data so the privacy of the data is violated. Alternatively, although the content of the data itself might remain secure, intruders could read and analyze the clear text source and destination identifiers attached to a message for routing purposes. They also could examine the lengths and frequency of messages that are transmitted. These latter attacks are known as *traffic analysis.* They can provide an intruder with important information about messages being transmitted. For example, analysis of source and destination identifiers can provide insights into troop movements in a military application, or they could provide sales information in a commercial application. Similarly, the lengths and frequency of messages could indicate the types of messages being transmitted.

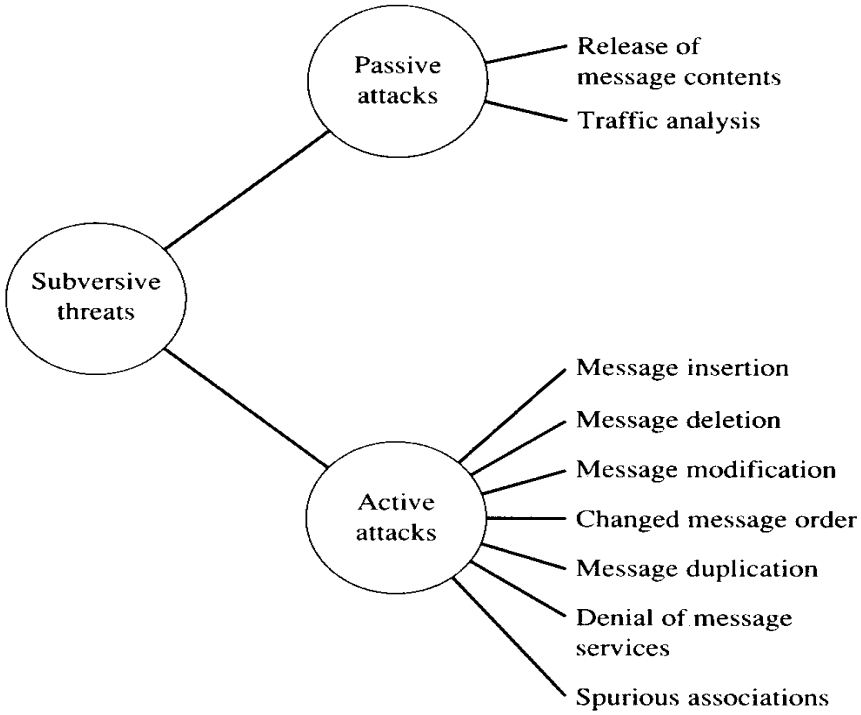


FIGURE 2-1 Subversive threats to the communication subsystem.

There are seven types of *active* attack:

1. Intruders could *insert* a message in the message stream being transmitted. For example, in an electronic funds transfer system (EFTS), they could add a deposit transaction for their account to the message stream being transmitted.

2. Intruders could *delete* a message being transmitted. For example, they could remove an account withdrawal transaction from the message stream being transmitted.

3. Intruders could *modify* the contents of a message being transmitted. For example, they could increase the amount field in a deposit transaction.

4. Intruders could *alter the order* of messages in a message stream. For example, they could change the sequence of deposit and withdrawal transactions to affect penalties incurred or interest charged or earned on their account.

5. Intruders could *duplicate* messages in a message stream. For example, they could copy deposit transactions for their account.

6. Intruders could *deny* message services between a sender and a receiver by corrupting (jamming), discarding, or delaying messages. This attack is similar to a message deletion attack. Message deletion is a transient attack on an established association between a sender and a receiver, however, whereas an attack that denies message services prevents the association from being established in the first place. It might be used by a competitor to severely impair the day-to-day operations of an organization.

7. Intruders could use techniques to establish *spurious associations* so they are regarded as legitimate users of a system. For example, they could play back a handshaking sequence previously used by a legitimate user of the system. Chapter 10 discusses this type of attack as part of our consideration of boundary subsystem controls.

2.3 Physical component controls

One way to reduce expected losses in the communications subsystem is to choose physical components that have characteristics that make them reliable and that incorporate features or provide controls that mitigate the possible effects of exposures. The following subsections give an overview of how physical components can affect communication subsystem reliability.

2.3.1 Transmission Media

A transmission medium is a physical path along which a signal can be transported between a sender and a receiver. Figure 2-2 shows the various types of transmission media that can be used in the communications subsystem. With *bounded* (or guided) media, the signals are transported along an enclosed physical path. Bounded media comprise twisted-pair wire, coaxial cable, and optical fiber. With unbounded (unguided) media, the signals propagate via free-space emission rather than along an enclosed physical path. Unbounded media comprise terrestrial and satellite microwave, radio frequency, and infrared.

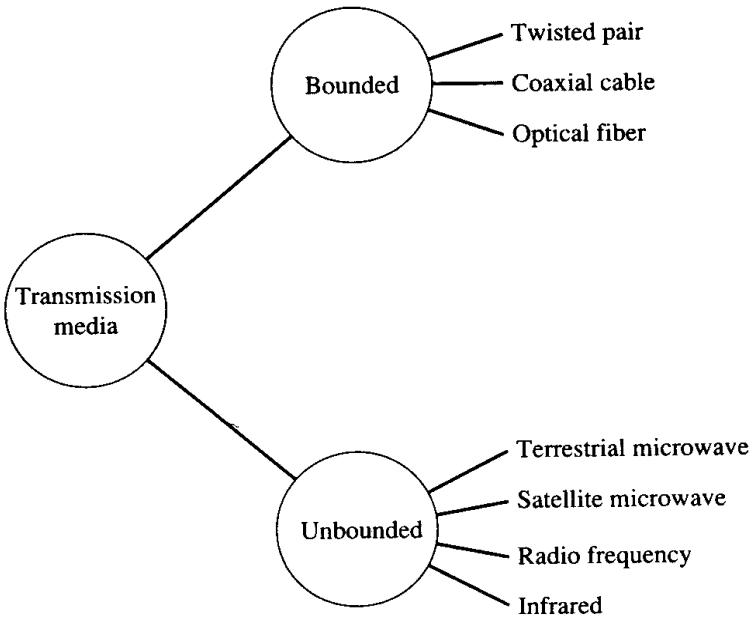


FIGURE 2-2 Types of transmission media

Tables 2-1a and 2-1b provide an overview of the capabilities of the various transmission media and their relative strengths and weaknesses in relation to the various exposures that can arise in the communication subsystem. *Twisted-pair* wire permits only a low rate of data transmission. Amplifiers for analog signals or repeaters for digital signals must be placed every few kilometers if data is to be transmitted over long distances. Unfortunately, amplifiers also increase distortion on the line. Moreover, by increasing the strength of the signal, both amplifiers and repeaters increase free-space emanations that can be picked up via an inductive wiretap. Indeed, wiretaps can be installed easily on twisted pair. Twisted pair is also highly susceptible to interference such as crosstalk and noise. These problems are offset to some extent, however, by the low cost of twisted pair.

Coaxial cable permits a moderate rate of data transmission over relatively short distances. If data is to be transmitted over long distances, amplifiers or repeaters must be installed more frequently than for twisted pair. Coaxial cable is moderately susceptible to various forms of interference, although less so than twisted-pair wire. It is relatively easy to install a wiretap on coaxial cable, although more difficult than with twisted pair. Coaxial cable is also more expensive to install than twisted pair.

Optical fiber permits very high rates of data transmission over relatively long distances before repeaters must be used. Although signals transmitted over optical fiber suffer from attenuation, nonetheless, they are immune to other forms of interference such as noise and crosstalk. Moreover, it is difficult to install a wiretap on optical fiber. Chao describes two methods of wiretapping that can be used, both of which are expensive. Under the *bending method,* the fiber is bent such that some light loss occurs at the

bend. Sensitive equipment is needed to pick up the light loss that results. Under the *insertion method,* the fiber is first broken.

TABLE 2-1a Characteristics of Bounded Transmission Media

Characteristic	*Twisted Pair*	*Coaxial Cable*	*Optical Fiber*
Data rate	Low	Moderate	High
Expense	Low	Moderate	Moderate
Distance	Moderate	Low–Moderate	High
Line of sight	No	No	No
Interference	High susceptibility	Moderate susceptibility	Low susceptibility
Wiretapping	High susceptibility	Moderate susceptibility	Low susceptibility

TABLE 2-1b Characteristics of unbounded Transmission Media

Characteristic	*Terrestrial Microwave*	*Satellite Microwave*	*Radio Frequency*	*Infrared*
Data rate	Moderate	Moderate	Moderate	Low
Expense	Moderate	High	Moderate	Moderate
Distance	High	High	Moderate	Low
Line of sight	Yes	Yes	No	Yes
Interference	High susceptibility	High susceptibility	High susceptibility	High susceptibility
Wiretapping	High susceptibility	High susceptibility	High susceptibility	High susceptibility

A tapping device and transmitter with a light source are then inserted. Providing a continuous flow of data is maintained over the fiber, installation of the insertion wiretap will be detected because the link will go down for a short period.

Terrestrial microwave permits moderate rates of data transmission over relatively long distances. Line-of-sight transmission is required; however, thus, a microwave station is needed about every 40 kilometers because of the earth's curvature. Microwave transmission is highly susceptible to various forms of interference. For example, rain causes signal attenuation. It is also relatively easy to install a wiretap. The wiretap will break the line-of-sight transmission, however, and therefore it should be easy to detect.

Satellite microwave permits moderate rates of data transmission over long distances. Line-of-sight transmission is maintained by having the satellite orbit the earth so it remains stationary with respect to its earth stations (achieved by having the satellite at a height of approximately 35,800 kilometers). Like terrestrial microwave, satellite microwave is highly susceptible to interference, and it can be wiretapped easily. If point-to-point transmission is used, installation of the wiretap should be detected because the signal will be broken. If the satellite is operating in broadcast mode (point-to-multipoint transmission), however, the wiretap will not be detected.

Any earth station within the area of broadcast will be able to pick up the satellite's transmission.

Radio frequency permits moderate rates of data transmission over moderate distances. Like microwave transmission, it is highly susceptible to interference. Radio frequency is omni-directional, however; that is, the signal is transmitted in all directions. Thus, it is easy to wiretap, and the wiretap will not be detected.

Infrared permits moderate rates of data transmission over short distances. It is highly susceptible to interference, and it requires line-of-sight transmission. Like microwave and radio frequency, it is easy to wiretap. The wiretap will break the line-of-sight transmission, however, and therefore it should be easy to detect.

2.3.2 Communication Lines

The reliability of data transmission can be improved by choosing a private (leased) communication line rather than a public communication line. Public lines use the normal public switching exchange facilities. As a result, users often have no control over the lines allocated to them for data transmission purposes. In some cases, however, users can specify the characteristics of the lines they require. The switching center will then allocate them a line having those characteristics. For small amounts of data transmission (generally, less than a few hours per day), public lines are cheaper than private lines.

Private lines are lines that are dedicated to service a particular user. In terms of transmission reliability, they have two advantages. First, they allow higher rates of data transmission. Thus, they are better able to accommodate the overheads associated with controls that might be implemented over transmitted data (e.g., encryption controls). Second, private lines can be conditioned; that is, the carrier ensures the line has certain quality attributes. A conditioned line limits the amounts of attenuation, distortion, and noise that its users will encounter.

2.3.3 Modems

Computer hardware uses and generates discrete binary signals (Figure 2-3a). For transmission purposes, these signals are sometimes converted to analog signals (Figure 2-3b). The device that accomplishes this conversion is called a modem or data set.

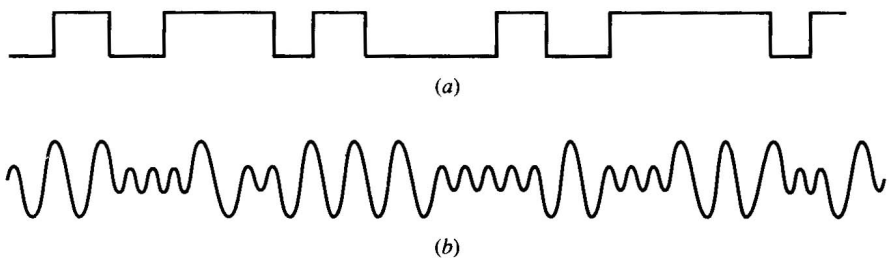


FIGURE 2-3(a) Digital signal (b) Analog signal

Modems undertake three other functions that affect the reliability of the communication subsystem. First, they increase the speed with which data can be transmitted over a communication line. They accomplish this objective by using some type of multiplexing technique (discussed subsequently). Higher rates of data transmission mean the overheads associated with controls have less impact. Higher rates of data transmission also mean that someone who successfully wiretaps a line gets access to more data.

Second, modems can reduce the number of line errors that arise through distortion if they use a process called *equalization.* If a modem has dynamic equalization capabilities, it will continuously measure the characteristics of a line and perform automatic adjustments for attenuation and delay distortion. Dynamic equalization is especially useful when public, unconditioned lines are used. Recall, that users have no prior knowledge of the characteristics of these lines.

Third, modems can reduce the number of line errors that arise through noise. To compensate for noise, a variable-speed modem will decrease the rate of data transmission as higher levels of noise are encountered. Recall that as transmission speeds increase, the effects of noise are more pronounced.

Modems work by varying either the amplitude, frequency, or phase of an analog signal to represent a digital signal (Figure *2-4*). Noise affects the performance of the three modulation methods differently. Phase modulation outperforms frequency modulation, which in turn outperforms amplitude modulation. The high-speed modems now in use sometimes employ a combination of methods to increase the speed of data transmission for example, amplitude and phase modulation.

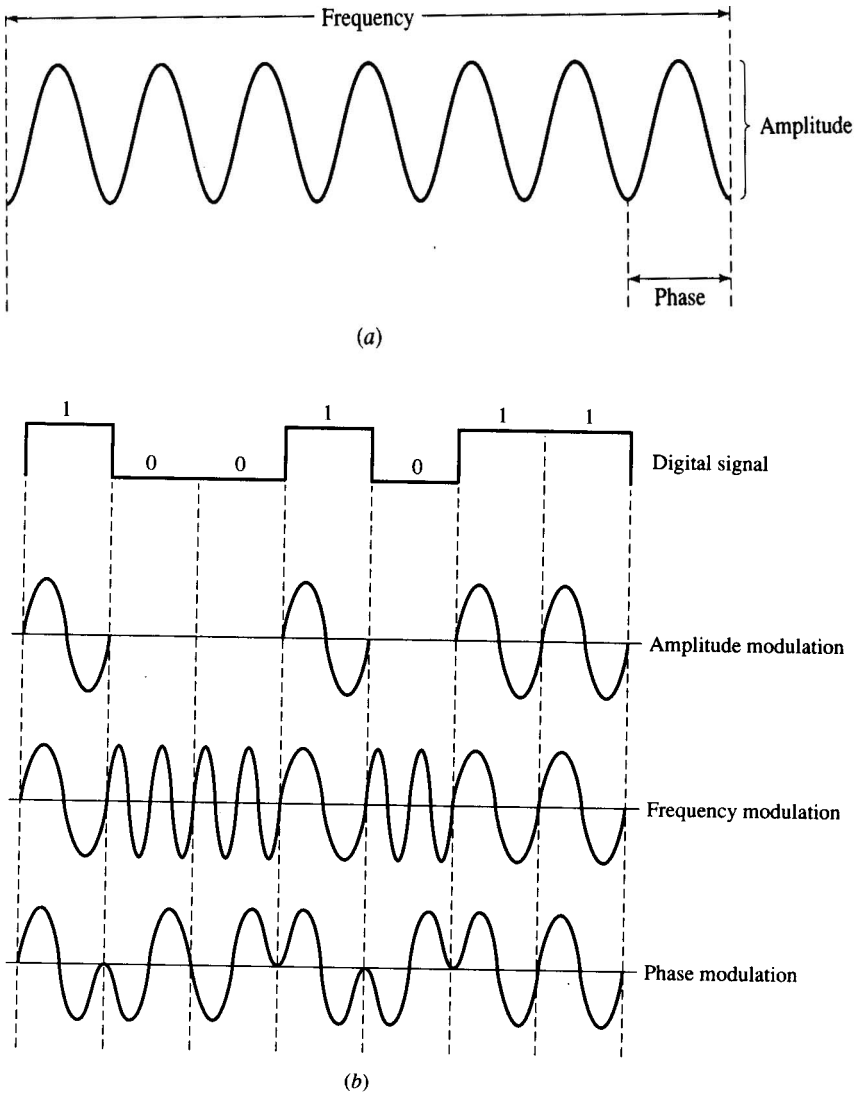


FIGURE 2-4 ((a) Wave form characteristics (b) Modulation techniques

2.3.4 Port-Protection Devices

Port-protection devices are used to mitigate exposures associated with dial-up access to a computer system. When users place a call to the system they seek to access, a connection is established with the port-protection device rather than the host system. The port-protection device then performs various security functions to authenticate users. Some examples of these security functions follow:

1. Users could be permitted to place calls to the host system only from authorized telephone numbers. When they dial the host system's telephone number, the port-protection device could disconnect them and then dial them back at their authorized number to ensure that they are calling from an authorized location.

2. A port-protection device could eliminate the telltale modem tone that autodialer routines can detect. It might respond to a call with a synthesized voice message or with silence as it awaits further user input.

3. Users could be required to provide passwords before the port-protection device will allow them access to the host system. The host system might in turn exercise its own user authentication functions.

4. Port-protection devices could maintain an audit trail of all successful and unsuccessful accesses to the host system. In addition, they might record the times when different activities occurred and the duration of each activity.

2.3.5 Multiplexors and Concentrators

Multiplexing and concentration techniques allow the bandwidth or capacity of a communication line to be used more effectively. The common objective is to share the use of a high-cost transmission line among many messages that arrive at the multiplexor or concentration point from multiple low-cost source lines.

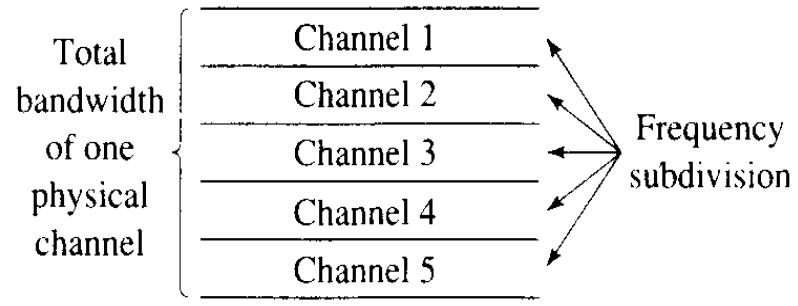
Multiplexing techniques use static channel derivation schemes to assign transmission capacity on a fixed, predetermined basis. Each data source shares a common transmission medium, but each has its own channel.

The two common multiplexing techniques used are frequency-division multiplexing and time-division multiplexing. The former divides a single bandwidth into several smaller bandwidths that are used as independent frequency channels (Figure 2-5a). The latter assigns small, fixed time slots to a user during which the user transmits the whole or part of a message (Figure 2-5b). Thus, channels are defined in terms of either a frequency band or a time slot.

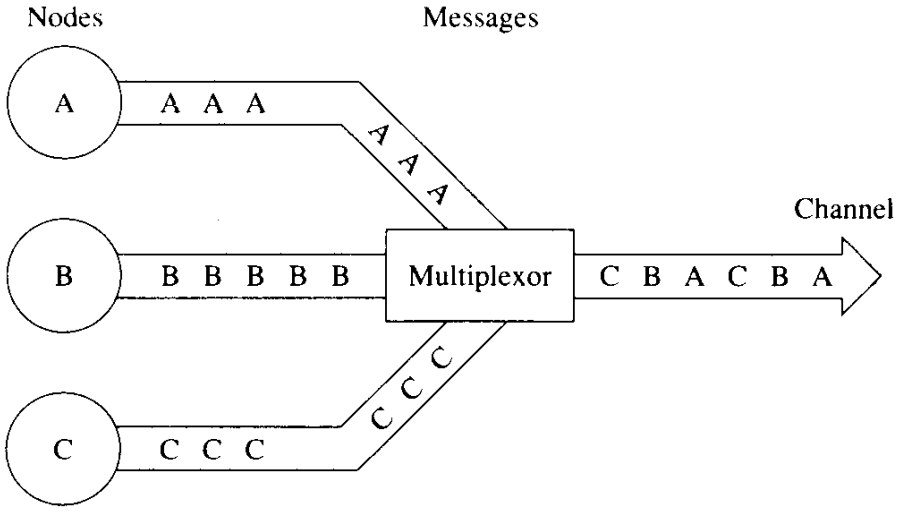
Two types of time-division multiplexing are used. *Synchronous* time-division multiplexing assigns time slots to each signal source on a round-robin basis. If the source has no data to transmit when its turn arrives, its corresponding time slot will be empty. *Statistical* or *asynchronous* multiplexing assigns time slots on a needs basis. Each signal source has an input buffer, and the multiplexor scans the input buffers collecting input data to fill a frame. The multiplexor sends the frame when it is full. In this light, statistical time-division multiplexing is more efficient than either frequency-division or synchronous time-division multiplexing because frequency bands and time slots are not wasted if a source has no data to send.

Concentration techniques use schemes whereby some number of input channels dynamically share a smaller number of output channels on a demand basis. Three common concentration techniques are message switching, packet switching, and line switching. In *message switching,* a complete message is sent to the concentration point and stored until a communication path can be established with the destination node. In *packet*

switching, a message is broken up into small, fixed-length packets. The packets are routed individually through the network depending on the availability of a channel for each packet. In *line switching* or *circuit switching,* a device establishes temporary connections between input channels and output channels where the number of input channels exceeds the number of output channels.



(*a*)



(*b*)

FIGURE 2-5(a) Frequency division multiplexing, (b) Time division multiplexing

Multiplexing and concentration techniques affect system reliability in several ways:

1. Both allow more efficient use to be made of available channel capacity. As a result, some channel capacity can often be used for backup purposes.

2. Concentration techniques can route a message over a different path if a particular channel fails.

3. Multiplexing and concentration functions often are incorporated into an intelligent front-end processor that performs other functions such as message validation and protocol conversion. These functions would otherwise be performed by the host processor, thereby increasing the workload of and reliance that must be placed on the host processor.

4. Both techniques help to protect data against subversive attacks. Wire tappers have greater difficulty disentangling the myriad of messages passing over a channel connected to a multiplexor or concentrator. Conversely, sophisticated intruders gain access to more data if they have suitable hardware and software and can determine the multiplexing or concentration techniques used.

5. Multiplexors and concentrators are critical components in a network. Thus, they should have a high mean-time-between-failure (MTBF).

2.4 Line error controls

Whenever data is transmitted over a communication line, recall that it can be received in error because of attenuation, distortion, or noise that occurs on the line. These errors must be detected and corrected. In the following subsections, we examine some major methods used to detect and correct transmission errors.

2.4.1 Error Detection

Line errors can be detected by either using a loop (echo) check or building some form of redundancy into the message transmitted. A *loop check* involves the receiver of a message sending back the message received to the sender (Figure 2-6). The sender checks the correctness of the message received by the receiver by comparing it with a stored copy of the message sent. If a difference exists, the message is retransmitted with suitable line protocol data to indicate the previous message received was in error. On some occasions, the message received might have been correct. The receiver's retransmission of the message back to the sender might have been corrupted, however, and hence a difference exists.

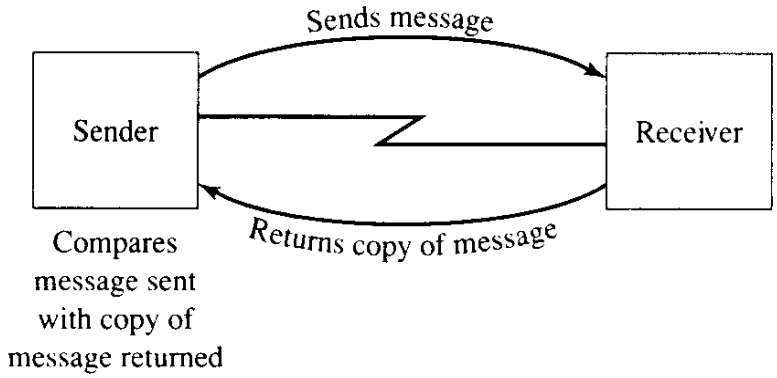


FIGURE 2-6 Loop check on communications line

Because a loop check at least halves the throughput on communication lines, normally it is used on full-duplex (simultaneous two-way communication) lines or where communication lines are short. If lines are short, the high protection afforded data transmission using a loop check could justify the costs of the extra channel capacity needed. On full-duplex lines, the return path is often underused anyway. In this light, it can be used productively for error detection purposes.

Redundancy involves attaching extra data to a message that will allow corrupted data to be detected. Two common forms of redundancy-based error detection methods are parity checks and cyclic redundancy checks.

Parity checking involves adding an extra bit to a string of bits. Figure 2-7 shows the use of both horizontal parity bits and vertical parity bits on a 10-character block of data. Each horizontal parity bit is used to detect whether a character in the block has been corrupted. Nevertheless, if a burst of noise on a line causes two bits in a character to flip from, say, zero to one, a horizontal parity check will not detect the error. A vertical or block parity check will detect the error, however.

With *cyclic redundancy checks* (CRCs), the block of data to be transmitted is treated as a binary number. This number is then divided by a prime binary number. The remainder is attached to the block to be transmitted. The receiver recalculates the remainder to check whether any data in the block has been corrupted.

Parity checks are employed when *asynchronous* data transmission is used that is, when data is transmitted one character at a time. CRCs can also be used with asynchronous transmission, but they tend to be used with *synchronous* transmission that is, where data is sent as a continuous stream of bits.

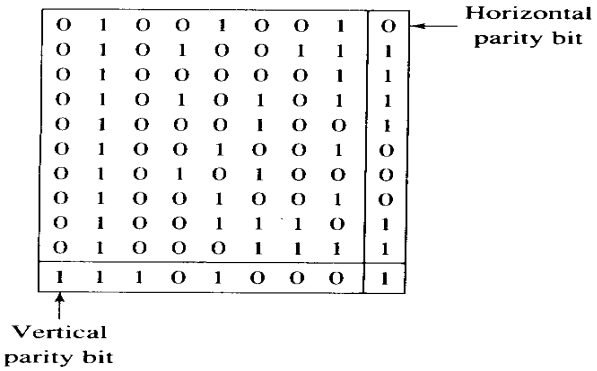


FIGURE 2-7 Horizontal and vertical odd parity check

2.4.2 Error Correction

When line errors have been detected, they must then be corrected. Two methods used are *(1)* forward error correcting codes, and *(2)* retransmission of data in error (backward error correction).

Forward error correcting codes enable line errors to be corrected at the receiving station. To determine what the correct data should be. redundant data must he added to the data transmitted. If line errors are infrequent, this redundant data can impose high overheads on the communication subsystem. Moreover, even with redundant data. there is always a risk that an attempted correction to an error will be carried out incorrectly. For these reasons, retransmission of erroneous data is often the error correction strategy chosen in preference to forward error correcting codes.

Nonetheless, forward error correcting codes have their place. For example, retransmission is costly when propagation times are long, such as in satellite transmission. Similarly, retransmission might be impractical in broadcast situations where multiple receivers exist. The sender would have to wait for multiple receivers to acknowledge correct receipt of the data. Moreover, only some receivers might receive corrupted data. Rebroadcasting the data would therefore impose overheads on receivers who received the data correctly.

With *retransmission,* the sender sends the data again if the receiver indicates the data has been received in error. An agreed-upon protocol is used to indicate correct or incorrect receipt of the message. In the ASCII ACK-NAK protocol, for example, an ACK signal is transmitted by the receiver if the message received is correct, and a NAK signal is transmitted by the receiver if the message received is incorrect. In some protocols, the sender wails for either an ACK or a NAK signal before transmitting the next message (stop and wait). In other cases, the sender continues to transmit messages while awaiting an ACK or NAK signal. If a NAK signal is received, the sender goes back to the message sent in error and retransmits messages from that point (go back N).

Noise also can corrupt the control characters used for retransmission in an error detection and correction system. An odd-even record count enables such errors to be detected. For example, consider a situation in which control characters are corrupted and two messages appear to the receiver to be a single message. Assume the control character for the first message was odd. The control character for the second message, an even number, has been corrupted. Thus, when the receiver identifies a third message having an. odd-numbered control character, it will recognize a message is missing and an error has occurred.

2.5 Flow controls

Flow controls are needed because two nodes in a network can differ in terms of the rate at which they can send. Receive, and process data. For example, a mainframe can transmit data to a microcomputer terminal. The microcomputer cannot display data on its screen at the same rate the data arrives from the mainframe. Moreover, the microcomputer will have limited buffer space. Thus it cannot continue to receive data from the mainframe and to store the data in its buffer pending display of the data on its screen. Flow controls will be used. Therefore, to prevent the mainframe swamping the microcomputer and. as a result, data being lost.

The simplest form of flow control is *stop-and-wait* flow control. Using this approach, the sender transmits a frame of data. When the receiver is ready

to accept another frame, it transmits an acknowledgment to the sender. The sender will not transmit another frame until it receives an acknowledgment from the receiver. Thus, the receiver controls the rate at which data reaches it.

The stop-and-wait flow control protocol is inefficient because the communication channel remains unused for periods of time while the receiver is processing the frames received. For this reason, the *sliding-window* flow control approach has been developed. Using this approach, both the sender and receiver have buffers that hold multiple frames of data which allow them to overlap transmission and processing of data.

Figure 2-8 provides an overview of a sliding-window control protocol. Assume the sender and receiver both has windows of seven frames (the shaded boxes at time?). Think of the data to be sent and received, also, as an endless sequence of data that will be broken up and allocated to one of eight windows in a sending or receiving buffer. The protocol works as follows:

1. At time t_1, the sender sends frames F0 and F1 to the receiver.

2. At time t_2, the sender's window shrinks by two frames to indicate frames F0 and F1 have been sent.

3. At time t_3, *the* receiver receives frames F0 and F1. The receiver's window shrinks by two frames to indicate receipt of the frames.

4. At time t_4, the receiver sends an acknowledgment to the sender indicating readiness to receive frame F2 (thereby signaling that F0 and F1 have been received safely). It then expands its window by two frames.

5. At time t_5, the sender receives the acknowledgment from the receiver and expands its window by two frames. It then sends four frames F2, F3, F4, and F5 to the receiver.

6. At time t_6, the sender's window shrinks by four frames to indicate frames F2, F3, F4, and F5 have been sent.

7. At time t_7 the receiver receives frames F2, F3, F4, and F5. The receiver's window shrinks by four frames to indicate receipt of the frames.

And so the process goes on. At any time, the receiver can also send a receive-Not-Ready (RNR) frame. This type of frame acknowledges receipt of prior frames but forbids the sender from transmitting any more frames until a further instruction is issued.

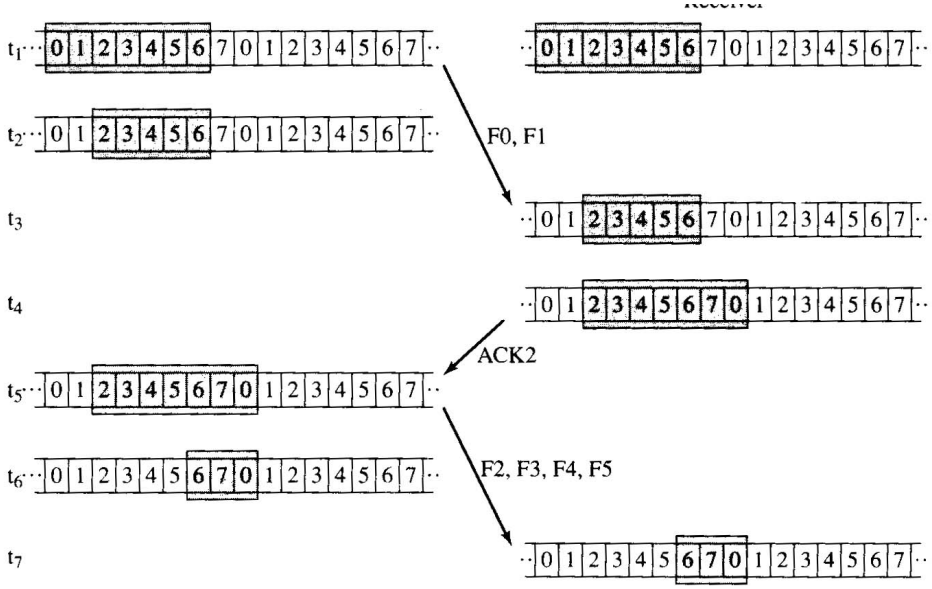


FIGURE 2-8 Sliding-window flow control

2.6 Link controls

In wide area networks, line error control and flow control are important functions in the component that manages the link between two nodes in a network. The way these link-management components operate is specified via a protocol. Two common protocols that are used are the International Organization for Standardization's Higher-level Data Link Control (HDLC) protocol and IBM's Synchronous Data Link Control (SDLC) protocol. Both are packet protocols developed specifically for computer communications. A newer packet protocol is Asynchronous Transfer Mode (ATM), which has been designed to handle a wider variety of data rates and data types than HDLC and SDLC. It is used commonly with broadband integrated services digital networks (B-ISDN), which have been developed to support high-speed communication of all types of data for example, sound and video as well as computer data.

From an auditors' viewpoint, however, they can have increased confidence in the likelihood of data being transferred accurately and completely between two nodes in a wide area network if well-developed and well-accepted protocols like HDLC, SDLC, and ATM are being used. To the extent that little-known or idiosyncratic data link protocols are being used, auditors must expand their evidence-collection work to obtain assurance that data are being transferred accurately and completely between two nodes in a wide area network.

2.7 Topological controls

A communication network topology specifies the location of nodes within a network, the ways in which these nodes will be linked, and the data transmission capabilities of the links between the nodes. Specifying the optimum topology for a network can be a problem of immense complexity.

Consider some of the design constraints that must be taken into account. First, an overall cost ceiling will apply, perhaps expressed as a limit on the cost per bit of information to be transmitted. Second, throughput and response time constraints exist. Communication of messages between different points in the network must be achieved within a certain time. Third, availability and reliability constraints exist. The network must be available for use at any one time by a given number of users. If a component of the network fails, alternative routing of messages or alternative hardware and software might be needed.

In the following two subsections, we examine the topologies that are commonly used in local and wide area networks. We briefly examine the nature of each topology and their strengths and weaknesses from a controls viewpoint.

2.7.1 Local Area Network Topologies

Local area networks tend to have three characteristics: *(1)* they are privately owned networks; *(2)* they provide high-speed communication among nodes: and *(3)* they are confined to limited geographic areas (for example, a single floor or building or locations within a few kilometers of each other). They are implemented using four basic types of topologies: *(1)* bus topology, *(2)* tree topology, *(3)* ring topology, and *(4)* star topology. Hybrid topologies like the star-ring topology and the star-bus topology are also used.

Bus Topology

In a bus topology, nodes in the network are connected in parallel to a single communication line (Figure 2-9). Each node in the network is passive. A tap is used to transmit data onto and receive data from the bus. Data is broadcast in both directions along the bus. At each end of the bus is a terminator that absorbs a signal on the bus, thereby removing it from the bus. A bus is a *multipoint* topology because more than two nodes share the same communication line

Two types of bus are used. A *baseband bus* uses digital signaling. The signal consumes the entire bandwidth of the transmission medium. Thus, frequency division multiplexing is not possible. Transmission is bidirectional over short distances. Repeaters must be used to increase the length of the network. Each repeater joins different segments of the communication line. A baseband bus is susceptible to wiretapping because only a single signal is traversing the bus.

A *broadband bus* uses analog signaling. Thus, frequency division multiplexing can be used to support different types of traffic on the network. Transmission is unidirectional because the amplifiers used in broadband bus networks are unidirectional devices. Thus, an inbound and an outbound path are required in the network. This objective is achieved by using either two cables (one for each path) or using different frequencies on the same cable. One end of the bus is designated as the head end. The head end then either acts as a passive conductor between the inbound and outbound cable, or it converts inbound frequencies to outbound frequencies on a single cable. Broadband buses cover longer distances than baseband buses because analog signals suffer less from attenuation, distortion, and

noise than digital signals. A broadband bus is also less susceptible to wiretapping than a baseband bus because multiple signals are traversing the bus.

From the auditors' perspective, the following control considerations arise with a bus topology:

1. Relative to other topologies like the ring, a bus degrades the performance ot the transmission medium because the taps that connect each node to the bus introduce attenuation and distortion to the signal being transmitted.

2. Because the taps that connect each node to the network are passive, the network will not fail if a node fails. Thus, bus networks are fairly robust when node failures occur.

3. Because all nodes have access to traffic on the network, messages not intended for a particular node can be accessed either deliberately or accidentally by the node. Thus, controls must be implemented to protect the privacy of sensitive data (e.g., encryption controls).

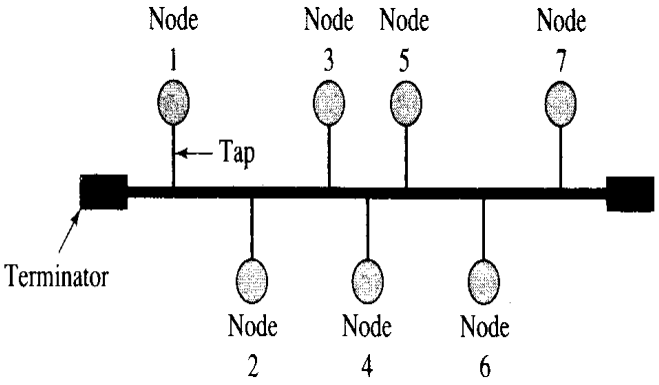


FIGURE 2-9 Bus network topology

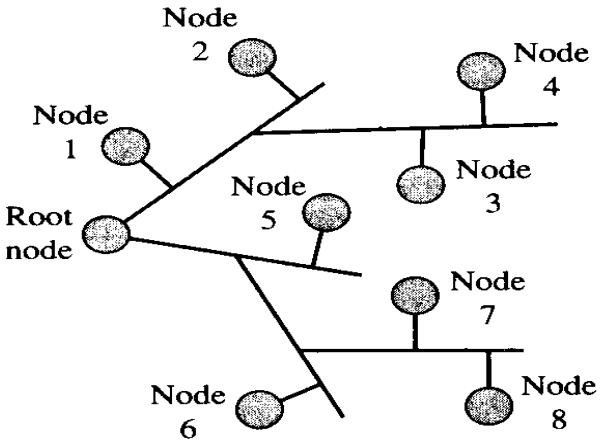


FIGURE 2-10 Tree network topology

Tree Topology

In a tree topology, nodes in the network are connected to a branching communication line that has no closed loops (Figure 12-10). In this regard, a tree topology is simply a generalization of a bus topology. As with a bus, messages are broadcast along the transmission medium. Moreover, as with a bus, each node uses a passive tap to broadcast data onto and receive data from the communication line.

It is difficult to propagate digital signals through the branching points that exist in a tree topology. Thus, tree topologies use analog signaling rather than digital signaling. Each node broadcasts messages in the direction of the root of the tree (which constitutes the head end). The root then propagates messages along the outbound path.

From the auditors' perspective, the control considerations that apply to a bus topology also apply to a tree topology.

Ring Topology

In a ring topology, nodes in the network are connected via repeaters to a communication line that is configured as a closed loop (Figure 2-11). The repeater is an active device: It inserts data onto a line, receives data from a line, and removes data from a line. In many ring networks, data is transmitted only in one direction on the ring (either clockwise or counterclockwise). A node often breaks a message up into packets of data and then inserts each packet on the ring. A ring is an example of a point-to-point (rather than a multipoint) topology. Each node is connected directly to another node; no intermediate node has to be traversed.

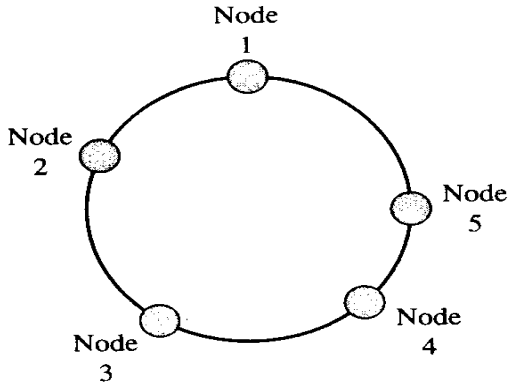


FIGURE 2-11. Ring network topology.

From the auditors' perspective, the following control considerations arise with a ring topology:

1. Unlike the taps used in a bus network, repeaters do not introduce attenuation and distortion to the signal being transmitted. Indeed, repeaters retransmit a clean signal after the signal has been received. Timing errors arise, however, as bits of data are transmitted around the network. These timing errors accumulate and effectively limit the number of repeaters that can be used in the network.

2. Because repeaters are active components in a ring topology, they will bring the network down if they fail. Repeaters might have a bypass mode, which is useful if their node is down.

3. Because all traffic on the network must be routed through each node's repeater, messages not intended for a particular node can be accessed either deliberately or accidentally by the node. As with bus networks, controls must be implemented to protect the privacy of sensitive data (e.g., encryption controls).

Star Topology

In a star topology, nodes in the network are connected in a point-to-point configuration to a central hub (Figure 2-12). The central hub can act as a switch. It can route messages from one outlying node to another outlying node. Alternatively, it can broadcast messages from one node to all other nodes or some subset of nodes.

From the auditors' perspective, the following control considerations arise with a star topology:

1. The reliability of the central hub is critical in a star network. If the central hub fails, the entire network will be brought down.

2. Failure in an outlying node or in a communication line linking a node to the hub has only a limited effect on the network. The remaining nodes can still transmit data to each other.

3. The security of the central hub is critical. All data must be transmitted through the hub. Therefore, compromise of the hub can mean all messages are compromised.

4. Servicing and maintenance of a star network are relatively easy. Diagnosis of problems can be performed from the central hub, and faults usually can be located quickly.

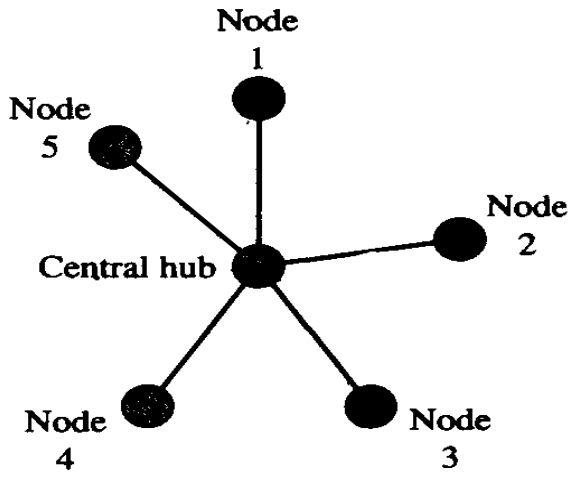


FIGURE2-12.Star network topology

Hybrid Topologies

Various types of hybrid topologies are also used in local area networks. For example, in the star-bus topology, nodes are connected via relatively long communication lines to a short bus that is usually housed in a wiring closet. Star-bus networks can be expanded easily, simply by connecting another drop cable to the bus. They have the control advantages and disadvantages of a bus. Nevertheless, they have an additional advantage because physically the short-length bus can be easily secured.

In the star-ring topology, nodes are connected via relatively long communication lines to a short-diameter ring. Again, this type of topology allows the network to be expanded relatively easily. From a control viewpoint, star-ring topologies have the advantages and disadvantages of a ring. They have an additional advantage, however, in that physically the short-diameter ring can be easily secured.

2.7.3 Wide Area Network Topologies

Wide area networks have the following characteristics:

(1) they often encompass components that are owned by other parties (e.g., a telephone company);

(2) they provide relatively low-speed communication among nodes; and

(3) they span large geographic areas.

With the exception of the bus topology, all other topologies that are used to implement local area networks can also be used to implement wide area networks. The most commonly used topology in a wide area network, however, is a *mesh* topology. In a mesh topology, conceptually every node in the network can have a point-to-point connection with every other node (a fully connected network). This topology is usually too expensive, however, and one node often must communicate with another node through intermediate nodes (Figure 2-13). A path between nodes is established using any of the concentration techniques discussed previously: message switching, packet switching, and line switching.

From a controls viewpoint, a mesh topology is inherently reliable because data can be routed via alternative paths through the network. If one path fails, the concentration methods used allow the data to be routed via another path. A major concern, however, is that reliance often must be placed on other parties to ensure the security, integrity, and reliability of the network. If, for example, a node that is owned and managed by another party is compromised, all data passing through this node is at risk. These concerns motivate the use of encryption controls in wide area networks.

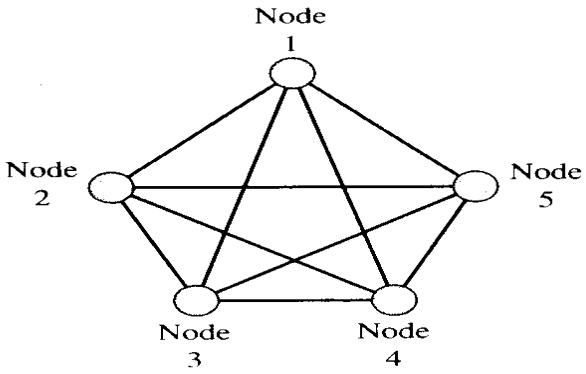


FIGURE 2-13 Mesh network topology

2.8 Channel access controls

Two different nodes in a network can compete to use a communication channel. Whenever the possibility of contention for the channel exists, some type of channel access control technique must be used. These techniques fall into two classes: polling methods and contention methods. The following subsections provide a brief description of each class of techniques.

2.8.1 Polling Methods

Polling (noncontention) techniques establish an order in which a node can gain access to channel capacity. There are two forms of polling: centralized polling and distributed polling. In *centralized polling,* one node within the network is designated as the control node or master node. This node takes responsibility for asking each other node in turn whether they wish to use the channel. For example, in the bus topology shown in Figure 2-14, the control node polls each other node according to a preset polling list: first node 1, then node 2, then node 3, then node 4, and so on, until it returns again to poll node 1. If a node has no message to send, the control node polls the next node on the list. If a node has a message to send, however, channel access is given to the node until its message or packet is sent. The next node on the polling list is then polled.

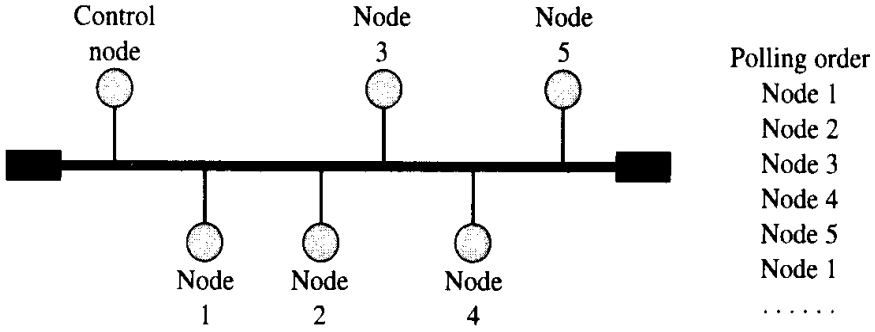


FIGURE 2-14. Centralized polling via a central node on a bus

In *distributed polling,* each node takes some responsibility for control over channel access. For example, a common form of distributed polling is *token passing.* A token is a special packet of information that traverses the channel. When a node wants to transmit data, it must first obtain the token and remove it from the channel. A common approach is to change a bit in the token so that it becomes a start-of-packet header. In Figure 2-15, for example, assume node 1 on the ring wants to send' a message to node 4. Node 1 will "grab" the token as soon as it passes, "attach" its message to the token, and insert it onto the ring. The address of the message will be read by each node as it passes along the ring. Node 4 will recognize that it is the intended recipient and read the message. It will not remove the message, however, from the ring. Instead, it will allow it to return to node 1 (the sender). Node 1 can then determine that the message has reached its intended destination. It will remove the message from the ring and then return the token to the ring.

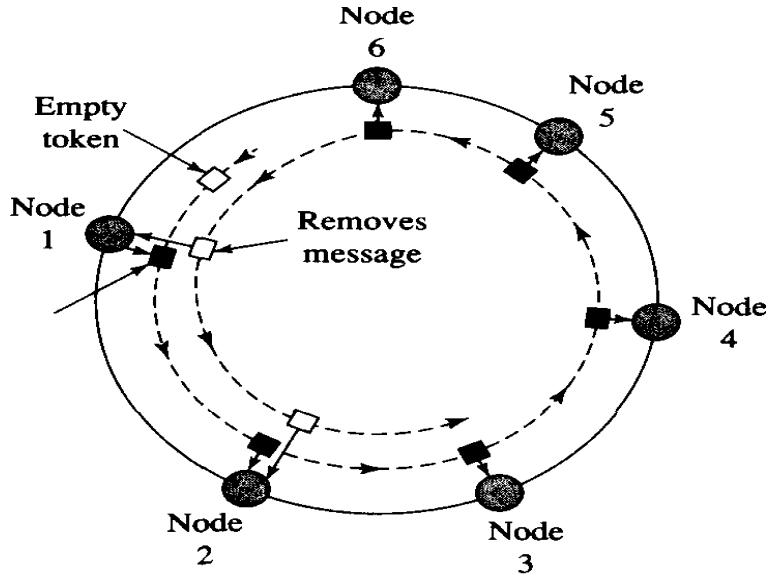


FIGURE 2-15. Distributed polling with token passing

Several types of problems can arise with token passing techniques:

1. A token can be corrupted as it traverses the communications line. As a result, nodes in the network that wish to send data will not be able to detect the token. Some type of protocol must exist whereby one of the nodes will introduce a new token on the channel if it does not detect a token within some time period. If that node fails to enter a token within some time period, another node might then enter a token onto the channel.

2. A node could fail to release the token after capturing it to read a message. Again, other nodes in the network that wish to send data will not be able to detect the token, and so a protocol must exist to reintroduce a token onto the channel.

3. The receiving node's address in a message could be corrupted. As a result, the receiving node might not detect a message intended for it.

The sending node must recognize that the token has returned without the message having been read by the intended receiver. The message must then be resent.

4. The sending node's address could be corrupted.

2.8.2 Contention Methods

Using contention methods, nodes in a network must compete with each other to gain access to a channel. Each node is given immediate right of access to the channel. Whether the node can use the channel successfully, however, depends on the actions of other nodes connected to the channel.

Although several different types of contention methods have been proposed, one commonly used with bus local area networks is called carrier sense multiple access with collision detection (CSMA/CD). With CSMA/CD, a node wishing to send a message first "listens" to the channel. If the channel is clear, it transmits the message to be sent. If another node also sends a message at the same time, however, a collision between the two messages will occur as they traverse the channel. To detect collisions, each node that sends a message must continue to listen to the channel. If a sending node "hears" a collision, it knows it must retransmit its message. To try to reduce the possibility of a further collision, a node will wait for some time interval before retransmitting its message. Different nodes will wait for different time intervals. If further collisions occur, nodes will wait longer time intervals before retransmitting their messages.

2.9 Controls over subversive threats

There are two types of control over subversive threats to the communication subsystem. The first seeks to establish *physical* barriers to the data traversing the subsystem. The second accepts that an intruder somehow will gain access to the data and seeks, therefore, to render the data useless when access occurs. We examine this type of control in the following subsections. Table 2-2 provides an overview of the controls discussed.

2.9.1 Link Encryption

Link encryption protects data traversing a communication channel connecting two nodes in a network (Figure 2-16). The sending node encrypts data it receives and then transmits the data in encrypted form to the receiving node. The receiving node subsequently decrypts the data, reads the destination address from the data, determines the next channel over which to transmit the data, and encrypts the data under the key that applies to the channel over which the data will next be sent.

With link encryption, the cryptographic key might be common to all nodes in the network in which case it is easy to establish a communication session between any two nodes, but it is difficult to protect the privacy of the key. Alternatively, each node must know the cryptographic keys of all other nodes with which it communicates in which case the keys are more secure, but key management is more difficult.

Link encryption reduces expected losses from traffic analysis. With link encryption, the message and its associated source and destination identifiers can be encrypted. Thus, a wire tapper has difficulty determining the identity of the sender and receiver of the message. In addition, frequency and length patterns in data can be masked by maintaining a continuous stream of cipher text between two nodes.

Type of Attack	*Control*
Release of message contents	Link encryption
	End-to-end encryption
Traffic analysis	Link encryption
Message insertion	Message sequence numbers
Message deletion	Message sequence numbers
Message modification	Stream ciphers
	Error propagation codes
	Message authentication codes
Changed message order	Message sequence numbers
Message duplication	Unique session identifiers
	Message sequence numbers
Denial of message services	Request-response mechanism
Spurious association	Secure authentication

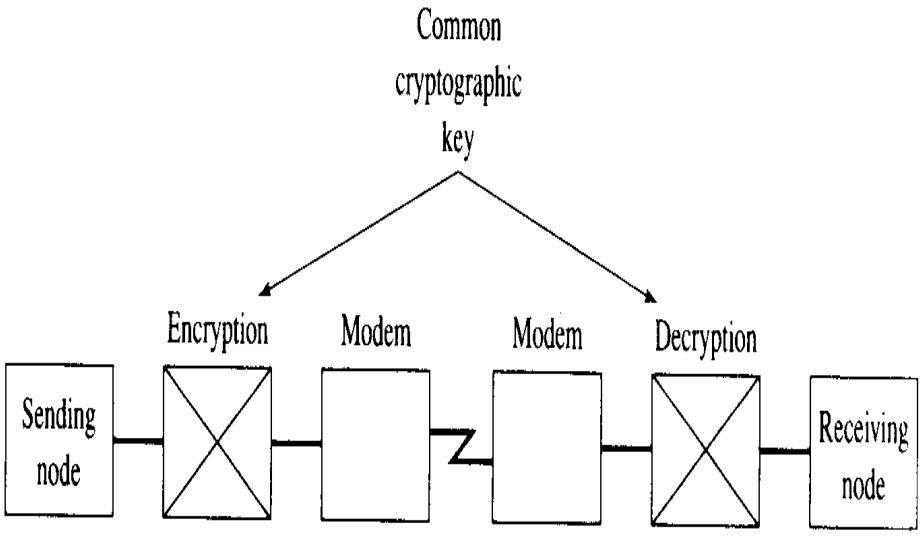


FIGURE 2-16. Link-encryption

Because each node must be able to decrypt a message to determine where it should be forwarded, link encryption cannot protect the integrity of data if a node in the network is subverted. In this light, encryption and decryption should be performed by a tamperproof cryptographic facility. Thus, even if intruders subvert a node in the network, the integrity of cryptographic operations is still protected.

2.9.2 End-to-End Encryption

Link encryption has several limitations:

1. If an intermediate node in the network is subverted, all traffic passing through the node is exposed. As a result, high costs might have to be incurred to protect the security of each node in the network. For example, security personnel might have to be present, physical barriers might have to be constructed, and regular security reviews might have to be undertaken.

2. Users of a public network might rely on link encryption to protect their data. In this light, the owners of the network could incur high insurance costs to protect themselves against damages resulting from security violations.

3. It can be difficult to work out a transfer-pricing scheme for allocating link encryption costs to users, particularly if some users argue that they do not need protection.

To help overcome these problems, end-to-end encryption can be used. End-to-end encryption protects the integrity of data passing between a sender and a receiver, independently of the nodes the data traverses. Thus, a cryptographic facility must be available to each sender and receiver because each now takes responsibility for implementing cryptographic protection. The sender en-crypts data before it is given to the network for transmission to the receiver. The data traverses each node and each communication channel in encrypted form. It is not decrypted until it reaches the receiver.

Although end-to-end encryption reduces expected losses from active or passive attacks when an intermediate node is subverted, it provides only limited protection against traffic analysis. Recall that with end-to-end encryption, intermediate nodes that the data traverses do not possess the key under which the data has been encrypted. If intermediate nodes are to send data over the correct route, therefore, the source and destination identifiers attached to a message must exist in the clear. In this light, link encryption can be used in conjunction .with end-to-end encryption to reduce exposures from traffic analysis.

2.9.3 Stream Ciphers

There are two types of ciphers: block ciphers and stream ciphers. With *block* ciphers, fixed-length blocks of cleartext are transformed under a constant fixed-length key (Figure 2-17). The Data Encryption Standard (DES) provides this mode of encryption via its electronic code book (ECB) mode. With *stream* ciphers, however, clear text is transformed on a bit-by-bit basis under the control of a stream of key bits. The stream can be generated in various ways. A widely used technique, however, is to make the key bit stream a function of an initialization value, an encryption key, and generated ciphertext. Figure 2-18 shows this method as implemented in the DES cipher block chaining (CBC) mode. The cleartext is first

partitioned into fixed-length blocks. Next, the first clear-text block is added (modulus 2) to this block. The result is enciphered, and the ciphertext produced is added (modulus 2) to the next cleartext block to be enciphered once more. The process continues iteratively.

Stream ciphers have two important characteristics. First, they make it more difficult to analyze patterns in ciphertext. In block mode, each enciphered block is independent of each other block. A cryptanalyst can examine character frequency patterns, which can lead to the cipher being broken. Stream ciphers, however, create interbit dependencies. Thus, patterns are masked from the cryptanalyst. Second, because interbit dependencies exist, changes to ciphertext propagate to subsequent ciphertext. If data is corrupted via a burst of noise on a line, at least some of the subsequent ciphertext will be unreadable. Similarly, if a wiretapper undertakes an active attack and modifies data in transit, at least some of the subsequent ciphertext will be affected. In this light, stream ciphers reduce expected losses from both active attacks and passive attacks.

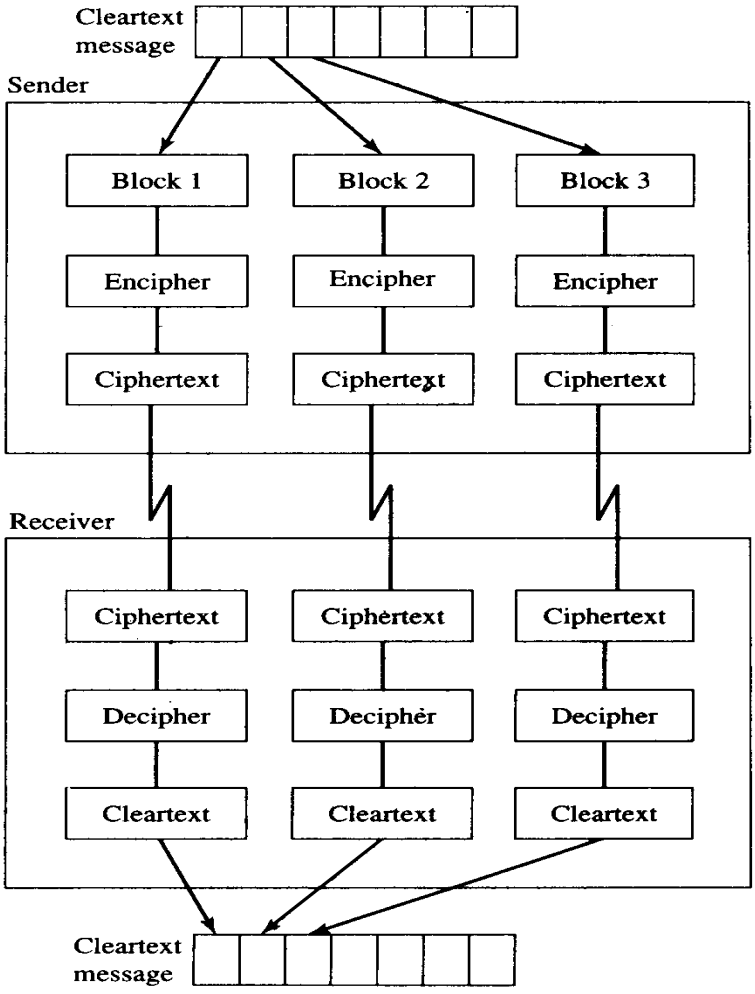


FIGURE 2-17 DES electronic code book (ECB) mode

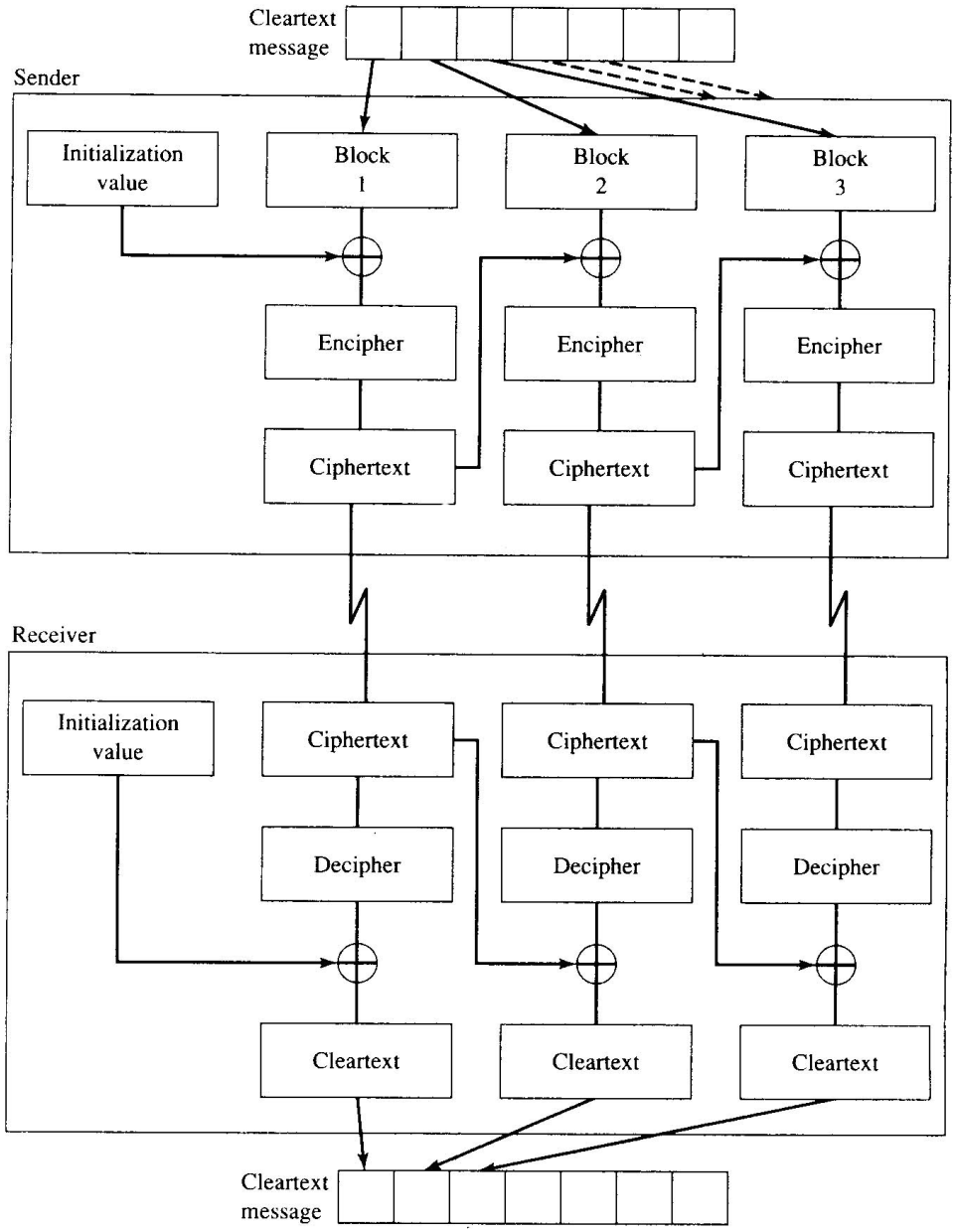


FIGURE 2-18 DES cipher block chaining (CBC) mode

2.9.4 Error Propagation Codes

Unfortunately, using stream ciphers alone is not sufficient to prevent all types of message modification. Even if messages are partitioned into fixed-length blocks and the CBC mode of encryption is used, cryptanalysts might still be able to make changes that will not be detected to the order of blocks within messages. This type of attack can also be undertaken successfully on other types of stream ciphers. The procedures to implement the attack are fairly complex, and we do not examine them here.

To protect the integrity of messages, a suitable error propagation code must be used. The code must be sensitive to the order of bits in a message so that a change to the order of blocks has a high probability of being detected. Several common error propagation codes, such as longitudinal parity checks, are unsuitable for this purpose because cryptanalysts can fairly easily determine changes to the ciphertext that will produce the same error detection code. Peterson and Weldon describe some error detection codes that are suitable because they are sensitive to bit order. The auditors' primary concern is to see that suitable error propagation codes have been chosen and implemented whenever attacks on message order could produce material losses.

2.9.5 Message Authentication Codes

In electronic funds transfer systems, a control used to identify changes to a message in transit is a message authentication code (MAC). The MAC is calculated by applying the DES algorithm and a secret key to selected data items in a message or to the entire message. Think of a MAC as an encrypted checksum calculated on the basis of some or all of the fields in a message. The MAC is then appended to the message and sent to the receiver, who recalculates the MAC on the basis of the message received to determine whether the calculated MAC arid the received MAC are equal. If the calculated MAC and the received MAC are not equal, the message has been altered in some way during transit. The transmitted message could be in the clear, or only selected data items in the message (such as the personal identification number) might be encrypted.

2.9.6 Message Sequence Numbers

Message sequence numbers are used to detect an attack on the order of messages that are transmitted between, a sender and a receiver. An intruder could delete messages from a stream of messages, change the order of messages in a stream, or duplicate legitimate messages. In each case the receiver does not obtain messages in the order generated by the sender.

If each message contains a sequence number and the order of sequence numbers is checked, these attacks will not be successful. It must be impossible, however, for the intruder to alter the sequence number in a message. Controls to prevent message modification have been discussed already: stream ciphers, error propagation codes, and message authentication codes. Furthermore, to prevent message duplication (playback), sequence numbers must not be reused during a communication session between a sender and a receiver. A unique identification number must be established for each communication session, and within this identification number each message sequence number must be unique.

2.9.7 Request-Response Mechanisms

A request-response mechanism is used to identify attacks by an intruder aimed at denying message services to a sender and a receiver. Recall that this type of attack is a form of message stream modification whereby the intruder deletes messages passing over a communication line or delays them for an extended period. If the parties to a communication session are not continuously communicating with each other, the receiver cannot detect

that a message should have arrived from the sender. The sender might realize that the receiver has not obtained the message because no acknowledgment has been returned by the receiver. In some cases, however, the sender might have no means of notifying the receiver that the communication channel has been broken. For example, in certain high-security applications, the sender might not be able to place a telephone call to the receiver to notify the receiver of the attack.

With a request-response mechanism, a timer is placed with the sender and receiver. The timer periodically triggers a control message from the sender. Because the timer at the receiver is synchronized with the sender, the receiver must respond to show that the communication link has not been broken. Providing that the timing signals can be generated with a pattern that is difficult to determine, the intruder will find it hard to undertake temporary undetected attacks that deny message service. In addition, the intruder must not be able to provide valid responses to the control messages. Otherwise, the sender will believe the channel is still open, and the receiver will be unaware that message services have been denied. Thus, those controls that establish the authenticity of the response must be applied to the request-response mechanism.

2.10 Internetworking controls

Internetworking is the process of connecting two or more communication networks together to allow the users of one network to communicate with the users of other networks. The networks connected to each other might or might not employ the same underlying hardware-software platform. In other words, internetworking might be based on either homogeneous networks or heterogeneous networks. The overall set of interconnected networks is called an *internet.* An individual network within an internet is called a *subnetwork.*

Three types of devices are used to connect subnetworks in an internet:

Device	***Functions***
Bridge	A bridge connects similar local area networks (e.g., one token ring network to another token ring network).
Router	A router performs all the functions of a bridge. In addition, it can connect heterogeneous local area networks (e.g., a bus network to a token ring network) and direct network traffic over the fastest channel between two nodes that reside in different subnetworks (e.g., by examining traffic patterns within a network and between different networks to determine channel availability).
Gateway	Gateways are the most complex of the three network connection devices. Their primary function is to perform protocol conversion to allow different types of communication architectures to communicate with one another. The gateway maps the functions performed in an application on one computer to the functions performed by a different application with similar functions on another computer.

Bridges, routers, and gateways perform several useful control functions. First, because they allow the total network to be broken up into several smaller networks, they improve the overall reliability of the network. Failure

in a node or communication line within a subnetwork, for example, will not disable all nodes in an internet. Second, for security reasons it might be desirable to keep different types of applications on different subnetworks. For example, high-exposure electronic funds transfer messages might be routed over a high-security, high-cost subnetwork. Low-exposure administrative messages, on the other hand, might be routed over a relatively insecure, low-cost subnetwork. Bridges, routers, and gateways might allow users of an internet to specify the subnetworks they wish their messages to traverse. Third, bridges, routers, and gateways may provide access control mechanisms to restrict access to subnetworks only to authorized users (Popalzai 1996). Not all users in a local area network, for example, may be allowed to access other subnetworks in an internet.

2.11 Communication architectures and controls

So far we have examined the functions and controls performed within the communication subsystem in a somewhat disjointed fashion. In an effort to provide an integrated view of these functions and controls, many people have developed coherent models of the communication subsystem. These models, or *architectures,* classify communication functions into a hierarchy of layers. Protocols for each layer are then defined, which are the set of syntactic, semantic, and timing rules governing the behavior of the components that provide the functions in each layer. Researchers have tried to show how various types of communication controls map onto the different layers in architecture.

Although a fair number of architectures have been proposed, three that have achieved prominence are the open-systems interconnection (OSI) architecture, IBM's system network architecture (SNA), and the transmission control protocol/internet protocol (TCP/IP) architecture. To illustrate the nature of these architectures and the placement of controls within them, we examine only the OSI model. Stallings provides a good discussion of the SNA and TCP/IP architectures.

The OSI architecture has been proposed by the International Organization for Standardization (ISO). The purpose of the architecture is to allow heterogeneous hardware/software platforms to communicate with one another across local area networks or wide area networks, providing they conform with the architecture. The architecture has seven layers of functions, each of which has associated controls:

Layer	*Name*	*Functions and Controls*
1	Physical	A hardware layer specifying both the mechanical features (transmission media and connectors) and electromagnetic features (voltage, signal strength, signalling method, amplification, modulation) of the connection between devices and the transmission medium. Receives data from some device and sends an unstructured bit stream over a transmission medium. The network topology (e.g., star) is part of the physical layer.
2	Data link	Primarily a hardware layer. Specifies channel access control method (e.g., HDLC, token ring). Ensures reliable transfer of data across the transmission medium. The bit stream is divided into blocks or frames of data. These blocks are then

Layer	*Name*	*Functions and Controls*
		subjected to synchronization, error control, and flow control. Link encryption used at this level.
3	Network	Chooses the physical route through which a message packet is sent through the network. Creates a virtual circuit for the upper layers so they are independent of the data transmission and switching technologies used to connect nodes. Establishes, maintains, and terminates connections between nodes. Ensures correct routing of data through the network.
4	Transport	Ensures reliable end-to-end transfer of data between user processes. Assembly and disassembly of message packets. Provides end-to-end error recovery and flow control. Multiplexing and end-to-end encryption undertaken at this level.
5	Session	Establishes, maintains, and terminates sessions (interactions) between user processes. Identification and authentication undertaken at this level. Might provide a checkpoint mechanism (see Chapter 13) so that recovery can be effected by retransmitting data from the last checkpoint.
6	Presentation	Controls how data appears on a screen. Transforms data to provide a standarized application interface. Encryption and data compression may also be performed at this level.
7	Application	Provides services to users—e.g., file sharing, file transfer, electronic mail. Database concurrency and deadlock controls may be performed at this level (see Chapter 14).

Figure 2-19 shows how two nodes using the OSI architecture communicate with each other via an intermediate node that also uses the OSI architecture. The sending node passes a message it wants to transmit down through the various layers. Hardware, firmware, or software perform the various functions assigned to each layer. When data reaches the physical layer, it is finally entered onto and sent over the transmission medium. Intermediate nodes will perform functions associated with layers 1-3 to ensure that the data is routed accurately, completely, and securely to its correct destination. The receiving node then passes the data it receives up through the various layers until it reappears to users at the application level. In effect, each layer in the sending node communicates with its peer layer in the receiving node (and vice versa) although the actual transmission of data occurs only at the physical layer level.

Auditors can organize their examination and evaluation of controls in the communication subsystem in terms of the various layers in the communication architecture they encounter. Unfortunately, the placement of controls within the various layers of a particular architecture is not always clear-cut. Indeed, the relationship of different controls to different layers is still a research issue with a number of communication architectures. In spite of these difficulties, however, the layers still provide a compelling way of thinking about controls in the communication subsystem.

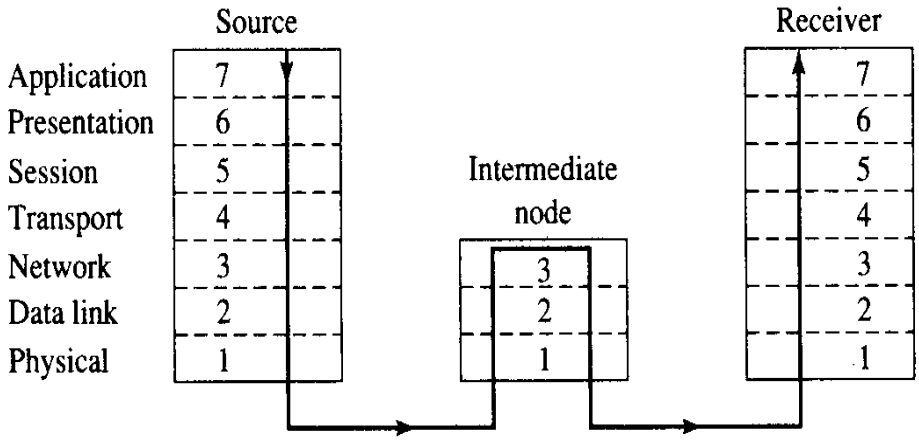


FIGURE 2-19 Transmission of data through the OSI layers.

2.12 Audit trail controls

The audit trail in the communication subsystem maintains the chronology of events from the time a sender dispatches a message to the time a receiver obtains the message. With the rapid growth in the use of data communications technology, the audit trail in the communication subsystem has become increasingly important. In an electronic data interchange system, for example, the absence of paper documents to support business transactions might mean that data contained in the audit trail is essential to the resolution of disputes that arise and the enforcement of contracts. Similarly, past messages dispatched via an e-mail system might be subpoenaed in the event of a dispute between parties to an electronic exchange of information.

2.12.1 Accounting Audit Trail

The accounting audit trail must allow a message to be traced through each node in a network. Some examples of data items that might be kept in the accounting audit trail follow:

1. Unique identifier of the source node,

2. Unique identifier of the person or process authorizing dispatch of the message,

3. Time and date at which message dispatched,

4. Message sequence number,

5. Unique identifier of each node in the network that the message traversed,

6. Time and date at which each node in the network was traversed by the message,

7. Unique identifier of the sink node,

8. Time and date at which the message was received by the sink node, and

9. Image of message received at each node traversed in the network.

Given that a message should not be changed as it traverses a node in the network, keeping all this information might seem pointless. Indeed, if a message traverses a public network or interchange network, the owner of the network might not be willing to maintain or supply the audit trail information. Nevertheless, the audit trail information is needed if *(1)* a message is lost in the network or *(2)* a node has been compromised or it has been malfunctioning and unwanted changes have occurred to the message. Nevertheless, what audit trail information should be kept and how long it should be kept as always is a cost-benefit decision.

2.12.2 Operations Audit Trail

The operations audit trail in the communication subsystem is especially important. The performance and, ultimately, the integrity of the network depend on the availability of comprehensive operations audit trail data. Using this data, network supervisors can identify where problem areas are occurring in the network. They can then reconfigure the network to mitigate the impact of these problems. Some examples of data items that might be kept in the operations audit trail follow:

1. Number of messages that have traversed each link,

2. Number of messages that have traversed each node,

3. Queue lengths at each node,

4. Number of errors occurring on each link or at each node,

5. Number of retransmissions that have occurred across each link,

6. Log of errors to identify locations and patterns of errors,

7. Log of system restarts, and

8. Message transit times between nodes and at nodes.

The availability of high-quality network control software is essential to network supervisors being able to make effective use of the operations audit trail. Although substantial operations audit trail data might be available, often it is not readily accessible or presented in a form that permits effective decision making. Good network control software will access the relevant operations audit trail data and provide reports that allow network supervisors to maintain or improve the performance of the network.

2.13 Existence controls

Recovering a communication network if it fails can be problematical. Some components might be complex, and determining the location and nature of a failure is often difficult. The status of the network upon failure also can be hard to assess. Message fragments might be dispersed throughout the network at various stages of processing. Ensuring complete and accurate

recovery of these in-flight messages can be difficult. It is also hard to provide backup for all network components. Some might be remote geographically. High costs could be incurred if redundant components are to be provided at these remote locations. Yet the network could be severely disrupted if long lead times are required to recover operations in these remote components.

Some backup and recovery controls that should be -implemented in a communication network are discussed earlier in this chapter when network reliability was considered: automatic line speed adjustment by modems based on differing noise levels, modems on private lines having automatic or semiautomatic dial-up capabilities for the public network, choice of a network topology that provides alternative routes between the source node and the sink node, and acquisition and use of high-quality network control software. Some additional backup and recovery controls follow:

1. Where possible, place redundant components (e.g., modems) and spare parts throughout the network.

2. Use equipment with in-built fault diagnosis capabilities.

3. Acquire high-quality test equipment.

4. Ensure adequate maintenance of hardware and software, especially at remote sites.

5. Ensure that adequate logging facilities exist for recovery purposes, especially where store-and-forward operations must be carried out in the network.

Because recovery can be a highly complex process that must be executed under severe time pressures, it is essential that well-trained, technically competent personnel operate the network. They must be provided with well-documented backup and recovery procedures either for a warm start (partial failure) or for a cold start (total failure). Given that multiple, physically dispersed components might have to be recovered in a coordinated way, a control site must exist for reporting all problems in the network and for managing personnel involved in the recovery process. Network backup and recovery procedures must be practiced regularly.

3. Database Controls

Structure

Objectives

After going through this lesson, you should be able to:

- understand to access controls;
- understand to Integrity controls
- understand to Application software controls
- understand to Concurrency controls
- understand to Cryptographic controls
- understand to File handling controls
- understand to Existence controls

3.1 Introduction

The database subsystem provides functions to define, create, modify, delete, and read data in an information system. Historically, the primary type of data maintained in the database subsystem has been *declarative* data that is, data that describes the static aspects of real-world objects and associations among these objects. For example, a payroll file and personnel file store information about the pay rates for each employee, the various positions within an organization, and the employees who have been assigned to each position. The database subsystem might also be used, however, to maintain *procedural* data that is, data that describes the dynamic aspects of real-world objects and the associations among these objects. For example, the database might contain a set of rules describing how an expert portfolio manager makes decisions about which stocks and bonds to choose for investment purposes. When both declarative and procedural data are stored, the database is sometimes called a *knowledge base* to reflect the greater "power" of the data maintained in the database subsystem.

The database subsystem is also being used increasingly to store *(1)* data about designs (e.g., manufacturing designs), in which the focus is design objects that can be composed or decomposed into other design objects, and *(2)* images, graphics, audio, and video, which can be used to support a multimedia application. In this light, substantial work is now being undertaken on *object-oriented* database management systems to support these types of applications. Moreover, with the emergence of huge databases and increasing use of decision support systems and executive information systems, there has been renewed interest in how databases should be structured to allow recognition of patterns among data, thereby facilitating knowledge discovery by decision makers. Huge databases that contain integrated data, detailed and summarized data, historical data, and metadata are sometimes called *data warehouses*. Databases that contain a selection of data from a data warehouse that is intended for a single function or department are called *data marts.* The process of recognizing patterns among data in data warehouses or data marts is sometimes called *data mining*.

Initially, the major components in the database subsystem were the application programs that defined, created, modified, and deleted data, the operating system that performed the basic input/output operations to move data to and from various storage media, the central processor and primary storage in which these activities were performed, and the storage media that maintained the permanent or semi-permanent copy of the data. For example, the activities previously performed by application programs and operating systems have been migrated to database management systems; special database machines have been developed to support the database subsystem; expert systems have been developed to support the processing of procedural data.

In this lesson we examine controls over the database subsystem. We begin by discussing the policies and mechanisms needed to prevent unauthorized access to and use of the database. Next we examine the various types of integrity constraints that a database management system should maintain over a database. We then discuss the various controls that can be used within application software to maintain the integrity of data, the controls that must be exercised to prevent integrity violations when multiple programs have concurrent access to data, the ways in which data privacy can be preserved via cryptographic controls in the database subsystem, and the ways in which files must be processed to prevent integrity violations. Finally, we examine audit trail controls and existence controls within the database subsystem.

3.2 access controls

Access controls in the database subsystem seek to prevent unauthorized access to and use of data. As with all subsystems, access controls are implemented by first specifying a security policy for the subsystem and then choosing an access control mechanism that will enforce the policy chosen.

We examined security policies that enforce *discretionary* access control and security policies that enforce *mandatory* access control. In the database subsystem, the former allow users to specify who can access data they own and what action privileges they have with respect to that data. The latter require a system administrator to assign security attributes to data that cannot be changed by database users. In the following two subsections, we examine how both types of security policy might be applied within the database subsystem. In the third subsection, we discuss some implementation issues relating to the access control mechanisms that are used in the database subsystem.

3.2.1 Discretionary Access Controls

In the database subsystem, discretionary access controls can vary considerably. For example, if a relational database management subsystem is used to support the database subsystem, a user may be authorized to do the following:

- Create a schema;
- Create, modify, or delete views associated with the schema;

- Create, modify, or delete relations associated with the schema;
- Create, modify, or delete tuples in relations associated with the schema; and
- Retrieve data from tuples in relations associated with the schema.

These privileges often are given to users who are designated as the "owners" of a particular schema and its associated views and relations. Nevertheless, some might be assigned to users even if they are *not* the owners of a schema and its associated views and relations. For example, nonowner users might be allowed to create relational tuples or to delete tuples.

If users are *not* the owners of a schema and its associated views and relations, however, often they will be assigned more restricted privileges than those assigned to owners. These restrictions can be many and varied. Some important types of restrictions, however, are the following:

1. *Name-dependent restrictions.* Users either have access to a *named* data resource or they do not have access to the resource. If users have access to a data resource, the action privileges they have must also be specified.

2. content-dependent restrictions. Users are permitted or denied access to a data resource depending on its contents.

3. context-dependent restrictions. Users are permitted or denied access to a data resource depending on the context in which they are seeking access.

4. History-dependent restrictions. Users are permitted or denied access to a resource depending on the series of access to and actions they have under taken on data resource.

3.2.2 Mandatory Access Controls

A classification level might also be assigned to a record/relation as a whole. The value of the classification level assigned to each record/relation should be equal to the highest classification level assigned to a data item/attribute in the record/relation.

3.2.3 Some Implementation Issues

A major factor affecting the reliability of the access control mechanism is the extent to which it is located in a single component or multiple components. Unfortunately, practical constraints often dictate that the access control mechanism be distributed across several components. As the functions to be performed by the access control mechanism increase, the size and complexity of the kernel increase correspondingly. Thus, devising efficient implementations of the kernel is more difficult to achieve. Moreover, it becomes more difficult to verify the accuracy and completeness of the functions performed by the kernel and to maintain the security and integrity of the kernel.

3.3 Integrity controls

A good database management system will enforce various types of *integrity constraints* within the database subsystem. Integrity constraints are established to maintain the accuracy, completeness, and uniqueness of instances of the constructs used within the conceptual modelling or data modelling approach used to represent the real-world phenomena about which data is to be stored in the database subsystem.

3.3.1 Entity-Relationship Model Integrity Constraints

The fundamental constructs in the entity-relationship model are entities, relationships between entities, and attributes of entities. *Entities* constitute the basic types (classes) of things (objects) in the real world to be modelled. Within the database subsystem, the following integrity constraints might be applied to entities:

Integrity Constraint	*Explanation*
Uniqueness	Each instance of an entity must be unique.
Maximum cardinality	Specifies the maximum number of instances of an entity that can exist in the database.
Minimum cardinality	Specifies the minimum number of instances of an entity that can exist in the database.
Entity identifier	Specifies the attribute(s) whose value(s) uniquely identify each instance of an entity.
Value type of identifier	Specifies the allowed value types for the attributes that comprise an entity's identifier—e.g., real number, integer, alphanumeric string.
Value set of identifier	Specifies the allowed set of values for the attributes that comprise the entity's identifier.

In the entity-relationship model, *relationships* reflect that two or more entities are coupled in some way. For example, a relationship might be shown between a student entity and a university entity to indicate that students attend universities. Within the database subsystem, a major type of integrity constraint that applies to relationships is a *cardinality* constraint. A cardinality constraint specifies either *(1)* the *maximum* number of instances of an entity that can be associated with an instance of another entity (or tuple of instances of multiple entities) or *(2)* the *minimum* number of instances of an entity that can be associated with an instance of another entity (or tuple of instances of multiple entities) (Figure 3-1).

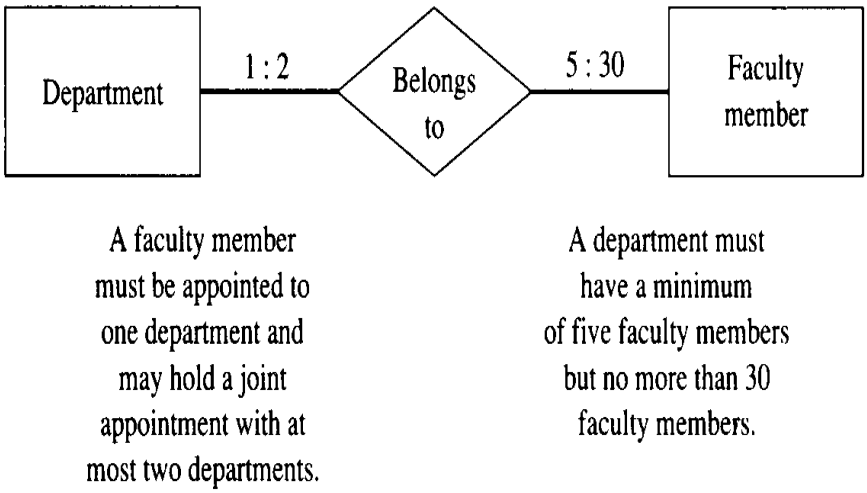


FIGURE 3-1 Cardinality integrity constraints on relationships within the entity-relationship model.

In the entity-relationship model, *attributes* reflect the properties possessed by entities. For example, two attributes of a person entity are age and gender. Within the database subsystem, the following integrity constraints might be applied to attributes:

Integrity Constraint	*Explanation*
Value type of attribute	Specifies the allowed value types for an attribute—e.g., real number, integer, alphanumeric string.
Value set of attribute	Specifies the allowed set of values for an attribute.
Transition law	Specifies the relationship between an attribute's previous value and its new value.

Relational Data Model Integrity Constraints

The fundamental construct in the relational data model is a relation. Within the database subsystem, the following integrity constraints might be applied to relations:

Integrity Constraint	*Explanation*
Key	Key constraints specify the candidate keys of a relation. Candidate key values must uniquely identify each tuple of a relation.
Entity	Entity integrity constraints are established to ensure that primary keys never have a null value.
Referential	Referential integrity constraints are established to maintain consistency among tuples in relations. If a tuple in a relation refers to data in another tuple of the relation or a tuple of another relation (via a foreign key), referential integrity contraints ensure that the referenced tuple must exist.

3.3.2 Object Data Model Integrity Constraints

The fundamental constructs in the object data model are objects and relationships among objects. *Objects* possess structural properties (attributes), which reflect the static characteristics of the object, and dynamic properties (methods or procedures), which reflect how the state of an object can change. Within the database subsystem, the following integrity constraints might be applied to the *structural properties* of objects:

Integrity Constraint	*Explanation*
Unique identifier	Each object must be unique. The database system can generate an object identifier that uniquely identifies the object throughout its life.
Unique key	Object keys are distinct from object identifiers. The former are system generated; the latter are user generated. Different constraints can apply to object keys. For example, keys might have to be unique within an object type or within all subtypes of a type. They might not have to be unique, however, across different object types.
Value type of attribute	Specifies the allowed value types for an attribute of an object—e.g., real number, integer, alphanumeric string, sets, lists, and arrays.

Integrity Constraint	*Explanation*
Value set of attribute	Specifies the allowed set of values for an attribute of an object. The values of an attribute might be defined procedurally (via a method) as a function of the values of other attributes of the object.
Types and inheritance	Ensures that an object of a subtype complies with all the integrity contraints associated with its supertype.

In the object data model, *dynamic properties* are the procedures that operate on objects. Dynamic properties facilitate encapsulation, whereby the structural characteristics of an object (or most of them) are kept hidden and only the procedures or methods (or at least a subset of them) are made public. Within the database subsystem, methods or procedures must comply with the syntactic and semantic rules of the language used to express them.

In the object data model, *relationships* among objects indicate that the property values of at least one of the objects depend on the property values of other objects in the relationship or that an object is a component of another object (aggregation or composition). Within the database subsystem, the following integrity constraints might be applied to relationships:

Integrity Constraint	*Explanation*
Referential	If one object refers to another object, the second object must exist and be of the correct type.
Composition	Specifies the actions to be undertaken upon insertion or deletion of objects that participate in composite relationships. For example, if an assembly is deleted from the database, all its subassemblies also must be deleted.
Cardinality	Specifies the minimum or maximum number of objects of particular types that can participate in a relationship.

3.4 Application software controls

As with the processing subsystem, the integrity of the database subsystem depends in part on controls that have been implemented in any application programs that use the database. Even though the database management system rather than the application software should directly access and update the database, nevertheless the database management system still depends on the application software to pass across a correct sequence of commands and update parameters and to take appropriate actions when certain types of exception conditions arise. Accordingly, the following

subsections describe various update and report protocols that might be implemented in application software to protect the integrity of the database.

3.4.1 Update Protocols

Update protocols in application software seek to ensure that changes to the database reflect changes to the real-world entities and associations between entities that data in the database is supposed to represent. We briefly examine some of the more important protocols in the following subsections.

3.4.2 Sequence Check Transaction and Master Files

In a batch update run, the transaction file is often sorted prior to the update of the master file or the tables in the database. In some cases, the master file or tables to be updated might also be sorted into a particular order. It might seem superfluous, therefore, for the update program to then check the sequence of the transaction file (and perhaps the master file or tables) as it processes each record. Nevertheless, sometimes situations arise that result in records on the transaction file (or master file) unexpectedly getting out of sequence. First, some "patching" of the file can occur because of a previous error. If the patching is done incorrectly, the file could get out of sequence. Second, an erroneous program could insert records in the incorrect sequence on the file. Third, on rare occasions a sort utility incorrectly sorts a file, or a hardware/system software error that corrupts the sequence of a file goes undetected. Fourth, undetected corruption of data might have occurred when the file was sent across a communication line.

3.4.3 Ensure All Records on Files Are Processed

If a master file is maintained in sequential order, correct end-of-file protocols must be followed in an update program to ensure records are not lost from either a transaction file or a master file. Common errors are to close the transaction file upon reaching the end of the master file or to close the master file upon reaching the end of the transaction file. In the former case, the transaction file might contain new records for insertion after the last record on the old master file. In the latter case, existing master file records could be lost because an end-of-file marker is placed at the point of closure. Designing and implementing correct end-of-file protocols can be an especially complex task if multiple sequential transaction files and multiple sequential master files are to be processed concurrently.

3.4.4 Process Multiple Transactions/or a Single Record in the Correct Order

Multiple transactions can occur for a single master record (tuple). For example, several sales orders plus a change-of-address transaction might have to be processed against a customer master record. The order in which transactions are processed against the master record can be important. Otherwise, several types of error can occur. For example, a customer might be billed at a wrong address, an employee might be paid after termination, or a person might receive welfare payments to which they are not entitled. Different types of transactions must be given transaction codes that result in their being sorted in the correct order before being processed against a master record.

3.4.5 Maintain a Suspense Account

Whenever monetary transactions must be processed against a master file (tables), the update program should maintain a suspense account. The suspense account is the repository for monetary transactions for which a matching master record cannot be found at the time an update is attempted. Mismatches can occur for several reasons; for example, an account number might be coded incorrectly in a transaction, a new account might not have been inserted correctly on the master file (table), or a transaction for a master record (tuple) might arrive before the master record (tuple) has been created. If monetary transactions for which a corresponding master record cannot be found are not charged to a suspense account, they can be lost because someone fails to correct the mismatch. Suspense accounts that have a nonzero balance provide a reminder that errors have occurred which still have to be corrected,

3.4.6 Report Protocols

Report protocols in application software have been designed to provide information to users of the database that will enable them to identify errors or irregularities that have occurred when the database has been updated. We briefly examine three such protocols.

3.4.7 Print Control Data for Internal Tables (Standing Data)

Many programs have internal tables that they use to perform various functions. For example, a payroll program might have an internal table of pay rates that it uses to calculate gross pays; a billing program might have an internal table of prices that it uses to prepare invoices; an electronic data interchange program might have a table that it uses to route orders to various suppliers; or an interest payment program might have a table of interest rates that it uses to pay customers interest on their bank accounts. Sometimes multiple versions of a table could be stored within the program. Different versions can take effect during different time periods, or perhaps a new version is prepared carefully over some time period to take effect after a certain date.

Maintaining the integrity of these tables is critical because the effects of errors in them can be substantial. For example, an error in a table of prices could mean a large number of customers are underbilled for merchandise they have received. It might be too costly to recover the monies lost. Furthermore, an organization might not want the error to be known publicly because of an adverse reaction by shareholders, creditors, and so on. In this light, access controls over the tables should be implemented to prevent unauthorized changes to them. Moreover, any changes made to internal tables (e.g., updating a pay rate) should be checked carefully for authenticity, accuracy, and completeness. One report protocol for standing data, therefore, is to print out internal tables after they have been changed to allow the changes made to be evaluated for authenticity, accuracy, and completeness.

Even if no changes are made to standing data, internal tables might still be printed periodically. Unauthorized changes could have been made to a table, or the table might have been corrupted in some way. Alternatively, if the table is large, it might not be printed and instead some type of control

total (such as a hash total) could be calculated and reported. Users can then check this control total to determine whether it differs from the previous control total.

3.4.8 Print Run-to-Run Control Totals

Sometimes the execution of an application system involves running multiple programs that pass files between each other. (Chapter 13 discusses the need to calculate and print run-to-run control totals as a basis for identifying errors in the processing subsystem in these situations.) Run-to-run control totals are also a useful means of identifying errors or irregularities that occur in the database subsystem. For example, they could signal that a record (tuple) has been dropped erroneously from a master file (table) that has been updated.

3.4.9 Print Suspense Account Entries

As mentioned previously, monetary update transactions that mismatch a master record must be written to a suspense account. To ensure that these transactions are ultimately cleared to their correct accounts, a suspense account report must be prepared periodically showing the transactions that were posted to the suspense account. The mismatches must also be written to an error file and removed as they are corrected. The suspense account report should remind users that they must take action to clear the errors if they are not removed from the error file promptly.

3.5 Concurrency controls

Unfortunately, sharing data resources produces a new set of problems that must be handled by the database subsystem if data integrity is to be preserved. In the following subsections we examine the nature of these problems and the various strategies that can be used to overcome them. Note that these problems are general in nature. Although we focus on shared *data* resources, they arise with any resource that might be shared.

3.5.1 Nature of the Shared Data Resource Problem

The best way to understand the problems that arise when we share data resources is via an example. Consider, then, an inventory application in a company in which a sales clerk and a receiving clerk have online access to the inventory master file. Assume, also, that they have *concurrent* access to the file; in other words, both can access the inventory master file at the same time.

Figure 3-2 shows a time sequence of events that can occur which results in data integrity being violated. First, a supplier delivers 100 units of good XYZ. As a result, the receiving clerk accesses the record for XYZ to update it. Input/output routines then copy an image of the existing record into the receiving program's buffer. The program next commences to update the image of the record. At the same time a customer places an order for 175 units of XYZ. As a result, the sales clerk accesses the record for XYZ to update it. Input/output routines copy an image of the record into the program's buffer. In the meantime, the receiving program completes its update and writes its buffer image of the record to the master file. The sales

program also carries out its update and writes its buffer image to the master file. Instead of the inventory record showing 425 units, it shows only 325 units.

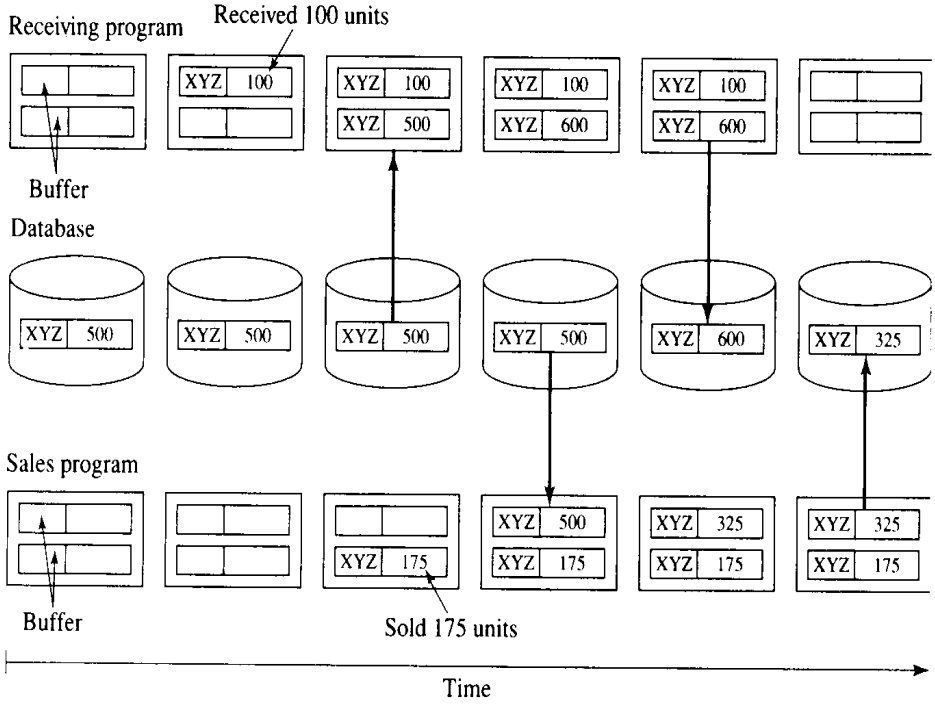


FIGURE 3-2 Concurrent processes as a threat to data integrity

Data integrity problems caused by concurrent processes are not confined to update programs. Read-only programs also can produce erroneous results if they operate concurrently with an update program. For example, a read-only program might be producing a trial balance for an accounts file. If an update program is concurrently posting debit and credit entries to the accounts, a situation might arise in which only one side (e.g., the debit) of a double-entry transaction is posted prior to the read-only program accessing the records to b updated. Thus, the trial balance will not balance.

The obvious solution to the data integrity problems caused by concurrer processes is to lock out one process from a data resource while it is being use by another process. Unfortunately, this solution leads to another set of problems. It can cause a system to come to a halt because of a situation called *deadlock* or the "deadly embrace."

3.5.2 The Problem of Deadlock

Figure 3-3 shows the problems that can arise when one process is allowed to lock out another process from a resource. At time *t,* process *P* acquires exclusive control of data resource 1, and process *Q* also acquires exclusive control of data resource 2. At time *t* + 1, process *P* makes an additional request for data resource 2, and process *Q* also made an additional request ft data resource 1. Neither process can continue until the other releases

content of the data resource that it has acquired at time t. A deadlock situation results.

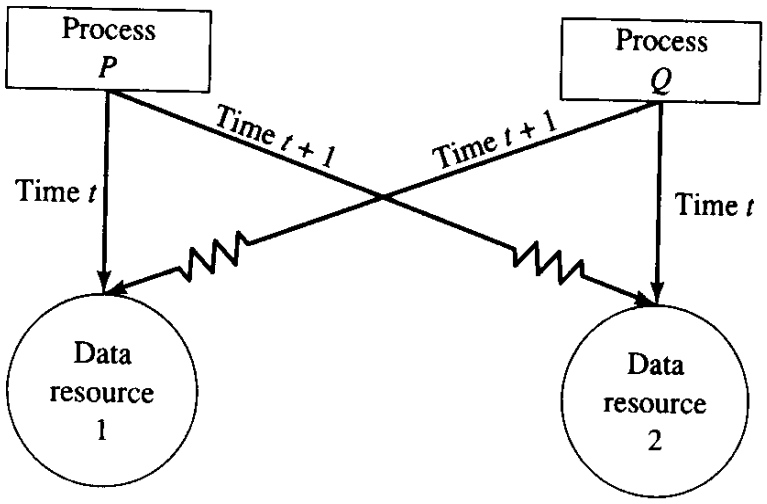


FIGURE 3-3 A deadlock situation.

The necessary and sufficient conditions for deadlock to occur follow:

Condition	*Explanation*
Lockout	A process can exclude another process from using a resource. Note that a read-only process might not wish to exclude other read-only processes. It might want to exclude other update processes, however.
Concurrency	Two or more processes can compete concurrently for exclusive control of two or more resources.
Additional request	While holding exclusive control of a resource, a process can request exclusive control of another resource.
No preemption	One process cannot force another process to release a resource prior to the process finishing with the resource.
Circular wait	A circular chain of processes exists. Each process in the chain holds a resource needed by the next process in the chain.

3.5.3 Solutions to Deadlock

How can a deadlock situation be resolved? At first thought we might believe the situation in Figure 3-3 can be overcome by simply forcing either process *P* or process *Q* to release the data resource over which it has exclusive control. This approach could indeed be a solution in some cases. Unfortunately, however, it does not always resolve the problem.

Consider the following simple example. Salesperson 1 receives a request from a customer for a certain set of parts say, 80 units of Part A and 90 units of Part B. The customer is unwilling to take the order unless all the parts requested can be supplied. Salesperson 2 receives a similar request from another customer say, 50 units of Part A and 100 units of Part B. Both salespersons initially query the database to determine whether sufficient inventory exists for all the parts requested (a read-only process). Recognizing that inventory could be depleted in the meantime because of other orders, they both commence to place their orders.

Figure 3-4 shows the situation that could result. Assume part A and part B is required in both salespersons' orders. Salesperson 1 acquires exclusive control of part A's record first and decreases the existing stock of 100 units by 80 units. At the same time, salesperson 2 acquires exclusive control of part B's record and decreases the existing stock of 150 units by 100 units. At time *t* + 1 a deadlock situation results. Consider what would happen if salesperson 1's program was allowed to preempt salesperson 2's program. After accessing part B's record, salesperson 1's program would find only 50 units (150 - 100) of part B available because salesperson 2's program has already updated part B's record. Thus, salesperson 1's order would have to be canceled in its entirety because 90 units of part B are required.

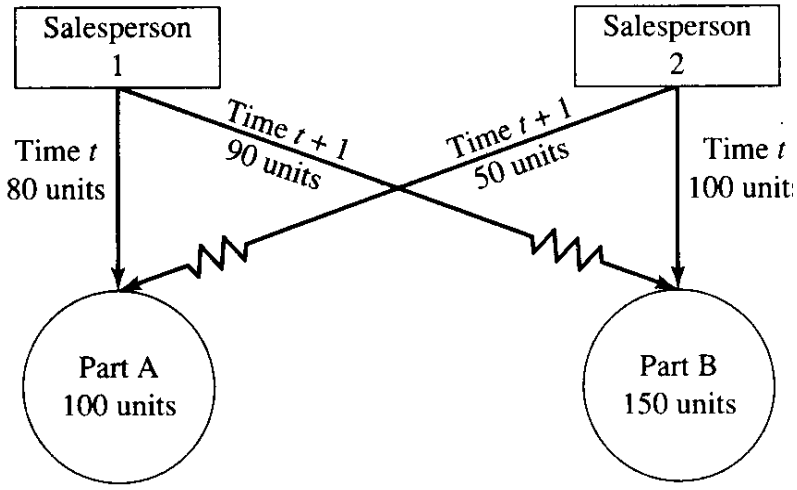


FIGURE 3-4. Inventory example of a deadlock

The same situation results if salesperson 2's program is allowed to preempt salesperson 1's program. Unless one program's updates are undone before the other program continues, both customer orders are lost, whereas only one order need be lost.

Problems arise because the database is in an inconsistent state when the preemption occurs. The updates of one program, therefore, need to be undone. Unfortunately, in some cases rolling back the actions that have been undertaken by a program can be difficult. Shipping notices, invoices, and so on might have been prepared and transmitted by the program that is preempted. The locations of these outputs have to be identified, and they

then have to be canceled. The degree of complexity associated with the preemption, therefore, is affected by how far the program needs to be rolled back. Moreover, a decision must also be made on which program should be preempted.

3.5.4 Preventing Deadlock

Over the years, a number of solutions have been proposed to resolve the problem of deadlock. The most widely accepted solution now, however, is called *two-phase locking.* It applies to a *transaction* that is being processed against the database. A transaction constitutes a sequence of interactions with the database that represents some meaningful unit of activity to a user. For example, in the context of the inventory application we examined previously, it would represent the sequence of operations needed to retrieve the relevant inventory records that satisfy the customer's request and to update them with the amounts ordered. Similarly, in the context of the trial-balance run discussed previously, it would represent the entire run from start to finish.

More precisely, however, a transaction must have four properties—the so-called ACID properties:

1. *Atomicity.* All actions taken by a transaction must be indivisible. Either all actions undertaken by a transaction are manifested in the current state of the database, or nothing is allowed to occur.

2. *Consistency.* A transaction must preserve the consistency of the database. The effects of a transaction are not reflected in the database until it *commits* its results. That is, all changes are first made in a temporary workspace until they can be written permanently, as an indivisible unit, to the database. Commitment is a two-phase process. During the first phase, the system writes the changes in the temporary workspace created for the transaction to some type of secure storage. If failure occurs during this phase, no harm has been done because the changes have not been applied to the database. During the second phase, the system copies the changes from secure storage to the database. If failure occurs during this phase, the new values of the database are recovered from secure storage. In either case, the transaction leaves the database in a consistent state.

3. *Isolation.* The events that occur within a transaction must be transparent to other transactions that are executing concurrently. In other words, no type of interference among transactions can be permitted.

4. *Durability.* When the results of a transaction have been committed, the system must guarantee that the changes survive any subsequent failure of the database. Existence controls, which we consider later in this chapter, are needed to achieve transaction durability.

Two-phase locking handles a transaction using the following protocol. First, before a transaction can read a data item, it must "own" a "read-lock" on the data item. Similarly, before a transaction can write a data item, it must own a "write-lock" on the data item. Second, different transactions are not allowed to own "conflicting" locks simultaneously. This rule means that two transactions can own read-locks on the same data item, but a read-lock

and a write-lock or two write-locks are not permitted to occur simultaneously. Recall that inconsistent results can be obtained if two processes concurrently read and write a data item or two processes concurrently write a data item. It does not matter, however, if two processes concurrently read a data item. Third, when a transaction releases ownership of a lock, it cannot obtain additional locks. Release of a lock gives another transaction the opportunity to obtain control over the data item, and the consistency of results can no longer be guaranteed. Thus, a transaction should commit its database changes before it releases its locks.

Two-phase locking, therefore, has a growing phase and a shrinking phase. During the *growing phase,* the transaction acquires locks without releasing locks. When the transaction releases a lock, it enters the *shrinking phase,* and it must proceed irrevocably to release all its locks. Locks could be released because *(1)* the transaction has committed its updates or *(2)* it has been unable to acquire all the locks it needs.

3.5.5 Distributed Database Concurrency Controls

When databases are distributed, their contents are stored at multiple sites. Various distribution strategies are used. At one extreme, a *replicated* copy of the database can be stored at all sites. At the other extreme, the database can be *fragmented* into nonoverlapping *partitions.* Each partition is then stored at exactly one site. Between these extremes are strategies where some data is replicated and stored at some subset of sites and other data is partitioned and stored at one site only.

Concurrency and deadlock problems can become a major threat to distributed database integrity unless the database management system has suitable controls. In the case of a *replicated* database, the system must somehow ensure that all versions of a data item are kept in a consistent state. In this light, some concurrency and deadlock strategies for replicated databases require that all instances of the data item needed must be locked before update operations can precede. In the case of *a partitioned* database, the location of the data item requested must be identified. Its lock" then must be activated.

The locking task for both replicated and partitioned databases conceptually might seem straightforward. Implementing it efficiently and reliably, however, is another matter. To illustrate how concurrency controls might work for a distributed database, consider the two-phase locking scheme examined previously for a centralized database. First, a two-phase *scheduler* must be constructed to process and to enforce the locking protocols. Next, in a *replicated* database, one of the following two strategies might be implemented:

1. Schedulers are replicated and stored with each version of the data item. If a read-lock is requested, the transaction need only request the lock at the most convenient scheduler. If a write-lock is requested, however, the transaction must request the lock of all replications of the scheduler for the data item needed. Alternatively, voting schemes might be used whereby a transaction is granted the lock if it acquires the lock from the majority of the data item's schedulers. It then notifies all of the data item's schedulers that it has been granted the lock. The voting method seeks to improve throughput, but it is more complex and error prone.

2. One version of the data item and its associated scheduler is designated as the primary copy. Before accessing a data item, a transaction must acquire the lock for the primary copy. The location of the primary copy is chosen to try to optimize system throughput. Transactions access a directory to determine the location of the primary copy. Alternatively, primary copies are all located at a central site. Under both approaches, difficulties arise if primary copies are lost or corrupted. Moreover, if primary copies are stored at a single site, they could all be lost if the site fails or is destroyed. For these reasons a second copy of the data item and its associated scheduler might be made, stored at a different site to the primary copy, and designated as the backup copy.

In a *partitioned* database, a transaction must acquire a read-lock or write-lock by first locating the scheduler for the data item requested. It must then activate the lock. Difficulties arise if the scheduler is lost or corrupted. As with a replicated database in which a primary-copy scheme is used, a second copy of the scheduler might be made and stored at another site as a backup copy.

Other strategies exist for handling concurrency and deadlock problems in distributed systems. Unfortunately, the area is complex. It could be difficult for auditors, therefore, to determine whether the concurrency controls implemented in a distributed database management system will maintain the integrity of data under all situations in which data resource conflicts arise.

3.6 Cryptographic controls

Cryptographic controls can also be used to protect the integrity of data stored in databases. The primary means of protecting stored data is block encryption. Recall that block encryption operates on individual blocks of data. It differs from stream encryption, in which the cryptographic value of one block of data depends on the cryptographic value of another block of data. Clearly, users who want to access a record or a data item in a record usually are unwilling to wait while the cryptographic mechanism decrypts all prior records or data items in the file. This outcome would occur under stream encryption. In short, stream encryption is useful for transferring entire files between two users, but block encryption should be used when users require access to only part of a file.

Data stored on portable storage media, such as tapes, diskettes, and cartridges, can be protected by implementing a secure encryption device in the device controllers for the media. Data is encrypted automatically each time it is written and decrypted automatically each time it is read. Although this type of encryption protects the privacy of data should the storage medium be stolen, it does not protect one user's data from another user because the cryptographic key used for encryption/decryption purposes is common to all users.

If little or no sharing of data among users occurs, individual users can protect their own files using a personal cryptographic key. They must present this key to the system when they wish to perform operations on their files.

This system is unsatisfactory, however, when data is shared. File owners must make their keys known to other users who require access to their file. As a result, encryption keys can become widely known, and the risk of key compromise increases.

Alternative cryptographic schemes have been devised when data must be shared. By way of illustration, however, consider the following scheme:

1. Each owner of a file is assigned *a file key* to perform cryptographic operations on the file.

2. A *secondary key* is assigned to encrypt/decrypt the file keys for the files owned by a particular user. Secondary keys create a protection domain that applies to a number of files, each of which has its own key. The file keys encrypted under the secondary key can be stored in the header records of the files to which they apply so they can be retrieved easily each time the file needs to be accessed. The file keys are secure because they are encrypted under the secondary key.

3. A *master key* is used to encrypt the secondary keys. Thus, the master key can be changed without having to reencrypt all data in the database. Only the secondary keys have to be decrypted and reencrypted.

4. To read and write a file, users must have access to the secondary key for the file so the file key in turn can be accessed and used for cryptographic purposes. Access to the secondary key can be protected using the standard types of access controls described in Chapter 10. Note the value of the secondary key is not revealed to users; they simply are permitted or denied access to it for cryptographic purposes.

Use of cryptographic controls in the database subsystem becomes more complex when the database is distributed. If the database is replicated, a decision must be made on whether the same keys will be maintained with each replication of the database. If a replica is lost or destroyed and the same keys are used, it is then relatively straightforward to make a copy of another replica to restore the lost replica. Moreover, it is also relatively easy to route a user transaction to another site if one site has a work overload and load balancing is being attempted. Several disadvantages arise, however, with this strategy:

(1) The keys must be distributed in a secure way; *(2)* they reside at more sites so the risk of compromise is higher; and *(3)* changes to keys at one site mean keys at all other sites must also be changed. Alternatively, if each site has its own set of keys, the keys will be more secure. It is more difficult to use replicas for backup purposes, however, and it is also more difficult to process transactions at sites other than the ones where they were initiated.

If the database is partitioned, in some 'cases data owned by a user could be located at multiple sites. If the same keys are assigned to a user across all sites where the user's data is located,' gaining access to the user's data is fairly straightforward. When a user transaction has been given the keys, it can access data at any site. Because the keys must be maintained at-multiple sites', however, the disadvantages described above again apply. Alternatively, if different sites have different keys, data at each site is more

secure, but higher processing overheads are incurred when transactions must access data at multiple sites.

3.7 File handling controls

File handling controls are used to prevent accidental destruction of data contained on a storage medium. They are exercised by hardware, software, and the operators or users who load and unload storage media (e.g., tapes,, diskettes, cartridges) used for the database, dumps of the database, transaction files,} work files, logs, and audit trails.

Several types of data can be stored in a file's header and trailer records so that a program can determine whether it is accessing the correct file:

Internal Data Item	*Nature*
Internal label	Specifies the name of the file, table, or database. Used by the program to check that it has accessed the correct file, table, or database.
Generation number	With removable storage media like tapes, diskettes, or cartridges, several versions of a file, table, or database can exist, all with the same name. Used by the program accessing the file, table, or database to check it is accessing the correct version.
Retention date	Prevents the contents of a file, table, or database being overwritten before a specified date.
Control totals	A record within each file, table, or database may contain control totals that can be checked. This record is updated at the end of each run, and the control totals are reported. On the next update run, users provide these control totals as parameters to the update program. The program then checks that they match the control totals on the file, table, or database that is to be updated as a basis for ensuring the correct file is being accessed.

Several hardware controls are also used to prevent accidental erasure of information on storage media. File protection rings are used to protect data on magnetic tapes. To enable data to be written to a tape, a plastic ring must be inserted into the recess at the back of the reel. If the ring is removed, data cannot be written to the tape. Similarly, disks can be protected by activating a readonly switch on a disk drive, and diskettes can be protected by sliding a plastic tab on the diskette so that the hole the tab covers is open. If multiple files are stored on a disk or diskette, individual files can also be locked by setting a flag to prevent erasure of data.

To assist users or operators in loading the correct files, external labels can be stuck on the outside casing of storage media. These labels can contain the name of the file, its creation date, and a code to indicate whether the file is a master file, a transaction file, a work file, or a backup file. External labels are not a substitute for internal labels, however, because labels can become detached from the storage media, or they might not be updated to reflect the current contents of the storage medium.

3.8 Audit trail controls

The audit trail in the database subsystem maintains the chronology of events that occur either to the database definition or the database itself. In many cases, the full set of events must be recorded: creations, modifications, deletions, and retrievals. Otherwise, it might be impossible to determine how the database definition or the database attained its current state, or who, via a retrieval transaction, relied on some past state of the database definition or the database.

3.8.1 Accounting Audit Trail

To maintain the accounting audit trail in an application system, the database subsystem must .undertake three functions. First, it must attach a unique time stamp to all transactions applied against the database definition or the database. This time stamp has two purposes: *(1)* It confirms a transaction ultimately reached the database definition or the database and *(2)* it identifies a transaction's unique position in the time series of events that has occurred to a data item in the database definition or the database.

Perhaps the most difficult problems encountered in supporting the audit trail arise from having to accommodate the effects of changes that occur within an application system. Consider the implications of the following types of changes on the audit trail:

1. A new data item is defined in the database definition and data collected to populate the database.

2. An existing data item is deleted from the database definition, and data is no longer collected for that data item.

3. The name used for a data item is changed.

4. A change of measurement scale occurs for a data item—for example, conversion from pounds to kilograms.

5. The coding system used for a data item changes—for example, conversion from a numeric to an alphanumeric code.

6. The key used to encrypt a data item is changed.

3.8.2 Operations Audit Trail

The operations audit trail in the database subsystem maintains the chronology of resource consumption events that affect the database definition or the database. On the basis of the operations audit trail, data administrators or database administrators can make two decisions. First, in light of response times or the amount of resources consumed when transactions are applied against the database, the need for database reorganization might become clear. Reorganization might involve establishing new access paths via indexes or pointers, clearing overflow areas, assigning data to faster storage devices, and so on. Second, re-source consumption data could indicate that the processes which apply transactions to the database definition or the database need to be

restructured or to be rewritten. For example, a database administrator might determine that a new database management system would better meet the needs of their organization, given the types of updates that occur to the database and the types of queries made on the database.

3.9 Existence controls

The whole or portion of a database can be lost (destroyed or corrupted) through five types of failure:

1. *Application program error.* An application program can update data incorrectly because it contains a bug. Usually only localized damage occurs to the database because the program updates only a small subset of data in the database.

2. *System software error.* System software, such as an operating system, database management system, telecommunications monitor, or utility program, could contain a bug. The bug might lead to erroneous updates of the database, corruption of data, or a system crash. Whether local or global damage occurs to the database depends on the nature of the bug.

3. *Hardware failure.* In spite of the high reliability of most hardware components, failure can still occur. The failure might be minor and transient, and as a result only localized damage occurs to the database. The failure might be serious and permanent, however, in which case extensive damage could occur to the database; for example, a disk crash destroys the contents of a disk.

4. *Procedural error.* An operator or user could make a procedural error that damages the database. For example, operators might undertake an incorrect action when recovering from a system crash, or a user might supply incorrect parameters to an update run. Whether the damage is local or global depends upon the nature of the error made.

5. *Environmental failure.* Environmental failure, such as flood, fire, or sabotage can occur. Often extensive damage to the database occurs. Off-site storage of files is essential to restoring the database after many types of environmental failure.

Existence controls in the database subsystem must restore the database in the event of loss. They encompass both a backup strategy and a recovery strategy. All *backup strategies* involve maintaining a prior version of the database and a log of transactions or changes to the database. If an update program creates a new *physical* version of a file, the previous version and the file of transactions used during the update can be used for backup purposes. If update occurs in place, however, periodically a dump of the database must be taken, and a log of changes to the database since the dump also must be maintained.

Recovery strategies take two forms. First, the current state of the database might have to be restored if the entire database or a portion of the database

is lost. This task involves a *rollforward operation* using a prior version or dump of the database and a log of transactions or changes that have occurred to the database since the dump was taken. Second, a prior state of the database might have to be restored because the current state of the database is invalid. This task involves a *rollback operation* to undo the updates that have caused the database to be corrupted. A log of changes to the database is used to restore the database to a prior, valid state.

In the following subsections, we examine the various forms of backup and recovery that can be used to restore a damaged or destroyed database. Auditors should have two concerns: *(1)* that a damaged or destroyed database can be restored in an authentic, accurate, complete, and timely way and *(2)* that the privacy of data is protected during all backup and recovery activities.

4. Out Put Controls

Structure

Objectives

After going through this lesson, you should be able to:

- understand how to Inference
- understand how to Batch output production and distribution controls
- understand to Batch report design controls
- understand to Audit trail controls

4.1 Introduction

The output subsystem provides functions that determine the content of data that will be provided to users, the ways data will be formatted and presented to users, and the ways data will be prepared for and routed to users. The major components of the output system are the software and personnel that determine the content, format, and timeliness of data to be provided to users, the various hardware devices used to present the formatted output data to users (e.g., printers, terminals, voice synthesizers), and the hardware, software, and personnel that route the output to users.

Many changes have occurred and are continuing to occur in the output subsystem. Much output that previously was produced as hard copy (printed output) is now produced as soft copy (displayed on a screen). With improvements in database technology, communications technology, and reporting software, users are better able to obtain the output they want directly rather than having to go through some intermediary. In this regard, printers are now often dispersed widely throughout organizations rather than being located in a single facility as they were in the past (the computer room). Output is now more varied—for example, relative to the 1980s, today much greater use is made of sound, video, and images as forms of output. Laser printers have become ubiquitous as the means of producing the reports, graphs, and drawings that previously were produced on impact printers and plotters. With the widespread availability of cheap, high-density storage devices such as CD-ROMs, greater use is now made of imaging software to read, store, and present images as output. Many organizations now seek to enable public access to information about them; for example, they establish pages on the World Wide Web that provide textual, video, image, and sound (multimedia) output to users who access these pages.

In this lesson we examine controls in the output subsystem. We begin by examining how inference controls can be used to filter the output that users are permitted to see. These controls are especially important when users are given access to statistical databases. We then discuss controls over the production and distribution of batch output. Next we consider how the design of batch reports can contribute to effective control over batch output. Subsequently we discuss controls over the production and distribution of online reports. Finally, we examine the audit trail and existence controls that should be implemented in the output subsystem.

4.2 Inference controls

The access control models examined in previous chapters permit or deny access to a data item based on the name of the data item, the content of the data item, or some characteristic of the query or time series of queries made on the data item. In some cases, however, it might be desirable to grant access to a data item but to restrict the type of information that can be derived from accessing the data item. This situation arises especially with statistical databases databases from which users can obtain only aggregate statistics rather than the values of individual data items. In statistical databases, sensitive and confidential data items, such as medical history data items, are maintained. Moreover, access to each data item is needed to provide statistical summaries of the information contained in the database. Users must not be able to deduce information about specific data item values, however, on the basis of a query.

Inference controls over statistical databases seek to prevent four types of compromise that can occur:

1. *Positive compromise.* Users determine that a person has a particular attribute value—for example, the person called John Doe is an alcoholic.

2. *Negative compromise.* Users determine that a person does *not* have a particular attribute value—for example, the person called John Doe is not an alcoholic.

3. *Exact compromise.* Users determine the precise value of an attribute possessed by a person—for example, the person called Mary Doe has a salary of exactly $120,000 per annum.

4. *Approximate compromise.* Users determine within some range the value of an attribute possessed by a person—for example, the person called Mary Doe has a salary in the range $100,000 to $140,000.

Note that a compromise must be either positive or negative, and it also must be either exact or approximate. Thus, there are four combinations: positive and exact (Doe's salary is $120,000); positive and approximate (Doe's salary is in the range $100,000 to $140,000); negative and exact (Doe's salary is not $120,000); and negative and approximate (Doe's salary is not in the range $100,000 to $140,000).

4.3 Batch output production and distribution controls

Batch output is output that is produced at some operations facility and subsequently distributed to or collected by the custodians or users of the output. It can take many forms for example, a hard-copy management control report;

pages containing tables, graphs, or images; negotiable instruments such as checks for distribution to employees, clients, or vendors; film or 35mm slides; a CD-ROM containing images and sound; microfiche containing document images; and a cartridge containing data for archival storage.

Production and distribution controls over batch output are established to ensure that accurate, complete, and timely output is provided only to authorized custodians or users. If the output is lost or corrupted, severe disruption can occur to the operations of the organization. For example, if customer bills are destroyed and the organization does not have backup, cash flow difficulties can arise. Likewise, if the privacy of output is violated, the organization can suffer losses in several ways: Unauthorized persons could use the information to gain access to the organization's information systems; confidential information, such as trade secrets, patents, marketing information, and credit information could be lost to a competitor; a criminal could blackmail the organization by threatening to expose confidential information about customers or clients.

We can structure our discussion on controls over batch output if we consider the different phases through which batch output must pass to be produced for and distributed to authorized custodians or users. As perhaps our most important example of batch output, consider batch reports. (We can use our discussion on batch reports also to illustrate controls that might be applied to other types of batch output.) Figure 4-1 shows the phases through which batch reports may pass in the output subsystem. Not all batch reports necessarily pass through each phase; for example, a report might be printed directly rather than queued or spooled. Moreover, what controls are exercised within each phase will depend on costs and benefits; more sensitive reports ought to be subjected to more controls. In the following subsections, however, we consider illustrative controls within each phase.

4.3.1 Stationery Supplies Storage Controls

Historically, when organizations used impact printers, they often had large amounts of preprinted stationery; for example, their invoice or customer statement stationery had a preprinted logo, address, and telephone number for the organization. It was more visually appealing to have these items preprinted (perhaps in color) rather than having them printed by an impact printer when output was produced. With widespread use of laser printers, however, use of preprinted stationery has declined. Many laser printers allow templates or masks of logos, addresses, telephone numbers, and so on, to be stored in them and printed (perhaps in color) on the plain-paper stationery they use. Alternatively, these templates or masks can be stored in programs (e.g., word processing programs) and printed on laser printers.

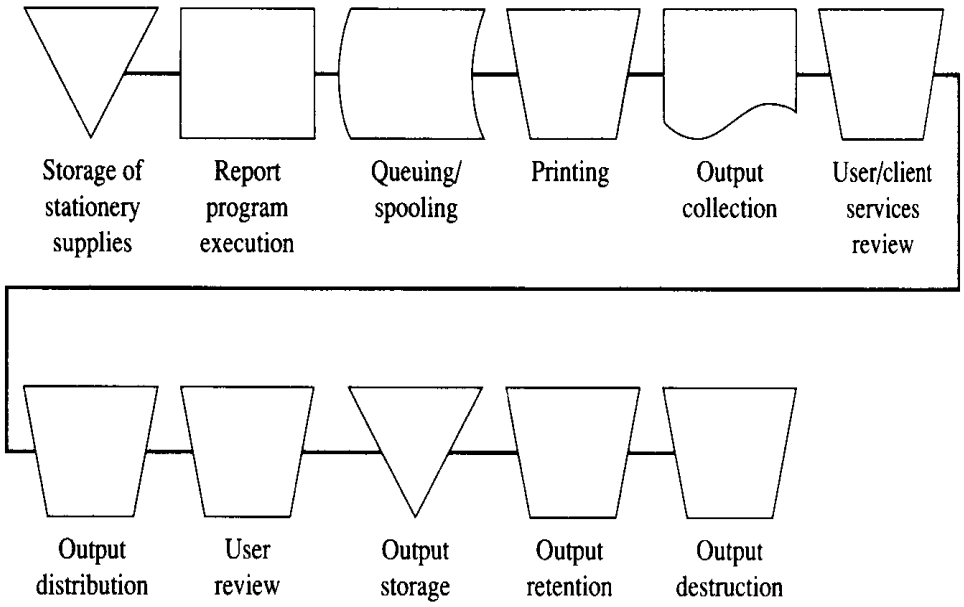


FIGURE 4-1 Stages in the production and distribution of batch output.

For several reasons, however, many types of preprinted stationery are still used. First, for some applications the preprinted features or the special paper on which the preprinting is done mitigate against forgeries. For example, negotiable instruments like checks are still used by some organizations, and they are likely to be prepared using preprinted stationery. Second, an organization might deem that preprinted stationery is still more visually appealing than printing constant output (e.g., the organization's logo) via a laser printer on plain-paper stationery. Various types of shading and dropout colors, for example, cannot be produced easily via laser printers. Third, printing speeds decrease and the costs of printing increase (e.g., because of increased consumption of toner) as larger amounts of information must be printed by a laser printer. In high-volume output applications, using preprinted stationery for output plight be faster and cheaper.

Whenever preprinted stationery is used, auditors should check to determine whether the organization exercises careful controls over the stationery. For example, the following controls might be used:

Control	*Explanation*
Stationery suppliers should produce preprinted stationery only under proper authorization and provide preprinted stationery only to authorized persons	Prevents unauthorized parties from obtaining copies of an organization's preprinted stationery to use improperly.
Maintain an inventory system for preprinted stationery	Helps account for all purchasing, receipt, and use of preprinted stationery.

Control	*Explanation*
Store preprinted stationery securely	Prevents unauthorized destruction or removal of stationery.
Control access to preprinted stationery	Only authorized persons should be able to gain direct access to stationery supplies.
Prenumber preprinted stationery	Facilitates control over use of preprinted stationery
Store signature stamps and preprinted stationery for negotiable instruments at separate physical locations	Reduces the exposure if either preprinted stationery for negotiable instruments or signature stamps are stolen.

In the absence of these controls, losses can arise. For example, check stationery might be stolen, and word processing software might then be used to print fraudulent details on the check. Moreover, authorized signatures can also be scanned and then subsequently printed on the check, perhaps to appear like a signature stamp or to give the impression of a valid written signature. Similarly, if an organization's customer database is stolen along with the organization's preprinted invoice stationery, a competitor can produce and distribute fraudulent invoices to the organization's customers to destroy the organization's goodwill.

4.3.2 Report Program Execution Controls

Auditors should have three concerns in relation to the execution of report programs. First, only authorized persons should be able to execute them. Otherwise, confidential data could be divulged. For example, a bank would want to restrict the execution of the program that prints personal identification numbers (PINs) for customers to only a few trusted employees. Likewise, most organizations would seek to restrict access to programs that print negotiable instruments like checks and authorization images like signatures to only a few trusted employees. During an audit, auditors should determine what report programs are sensitive. They should then check to see which users have access to these programs. They should also check whether high-quality identification and authentication

controls are in place and working to ensure that only authorized users access the programs.

Second, the action privileges assigned to authorized users of report programs should be appropriate to their needs. For example, an organization might wish to limit the number of copies of a report that a user can produce or to limit production of a report to certain times of the day or certain days within a month. Auditors should check to see whether the access control mechanism used by the organization can enforce these types of restrictions. If it can enforce this type of restriction, auditors should then evaluate the ways in which action privileges have been assigned to users.

Third, report programs that produce a large amount of output should include checkpoint/restart facilities. Recall from Chapter 13 that checkpoint/restart facilities can reduce the amount of work that has to be redone when some type of system failure occurs. From the viewpoint of our evaluating controls in the output subsystem, however, auditors should be concerned to see that checkpoint/restart facilities are not misused. For example, they should not be invoked to produce an unauthorized copy of a report. If auditors are concerned about this exposure, they can examine the operating system log for evidence of unauthorized use of checkpoint/restart facilities associated with sensitive report programs.

4.3.3 Queuing/Spooling/Printer File Controls

If a report program cannot write immediately to a printer, the output is queued or spooled. System software causes the report program to "think" it is writing to the printer when actually it is writing a printer file to disk storage. When the printer becomes available, spooling software reads the file and produces the report.

The presence of an intermediate file in the printing process leads to two control problems. First, printer files provide opportunities for unauthorized modifications to and copying of reports. For example, software can be used to change the value of a data item in a printer file, perhaps to prevent disclosure of some irregularity. Similarly, a copy of the printer file can be made and transmitted to an off-site location. Second, spooling software might be used inappropriately. For example, to recover from a printer malfunction, spooling software might allow operators to return to some prior intermediate point and to restart printing of a report. Unauthorized copies of the report might be produced in this way. Likewise, spooling software might allow operators to request more copies of a report than the number requested by the person who initiated printing of the report. Again, unauthorized copies of the report might be made.

Auditors must evaluate how well their client organizations achieve the following control objectives in relation to queuing or pooling of printer files:

1. The contents of printer files cannot be altered.

2. Unauthorized copies of printer files cannot be made.

3. Printer files are printed only once.

4. If copies of printer files are kept for backup and recovery purposes, they are not used to make unauthorized copies of reports.

4.3.4 Printing Controls

Controls over printing have three purposes: *(1)* to ensure that reports are printed on the correct printer; *(2)* to prevent unauthorized parties from scanning sensitive data that are printed on reports; and *(3)* to ensure that proper control is exercised over printing of negotiable forms or instruments.

Reports can be printed either intentionally or unintentionally on the wrong printer. If users have access to multiple printers, probably they will select which printer they wish to employ by making choices from a screen menu. Some printers that users can access might be secure because they are kept behind locked doors. Others might be insecure because they are located in public areas. Users who seek to perpetrate an irregularity could intentionally direct a sensitive report to a printer that is not in a secure location to provide a copy to an unauthorized user or to facilitate unauthorized removal of a copy themselves to an off-site location. Printing might also occur on the wrong printer, however, simply because *(1)* users forget to change their menu selection from a printer that is not in a secure location to one that is in a secure location or *(2) a* fault in the communications subsystem sends output to an incorrect device address.

Several steps might be undertaken to try to ensure that sensitive reports are printed only on a secure printer. First, users might be permitted to activate printing of sensitive reports from workstations that can access only secure printers. Second, users might be trained to check that they have selected the correct printer before they print sensitive reports. Third, software controls might be implemented to trap sensitive reports before they are printed on a printer that is not in a secure location

4.3.5 Report Collection Controls

When output has been produced, it should be secured to prevent loss or unauthorized removal, especially if the output contains negotiable instruments. For example, user/client services group employees might collect output reports, film, or cartridges and hold them pending collection by users. They should collect the output promptly and store it securely. Alternatively, if users direct output to an unattended device say, reports to a laser printer they should be responsible for prompt collection of the output, especially if the device is in a public area.

If user/client services group representatives have responsibility for collecting output, they should maintain records of the output they handle. For example, they should note the date and time when each output item was collected and the state of the output when it was received for example, whether all output was intact (see the next section). The identity of the user/client services group representative who collected the output should also be recorded.

Controls should exist to identify when output is not collected promptly and secured. For example, the manager in charge of information systems operations should have a system in place to indicate when any output has

not been collected promptly from a production facility. Similarly, someone should have responsibility for removing and securing output that has not been collected promptly from output devices that are in public areas.

4.3.6 User/Client Services Review Controls

Before output is distributed to users, a user/client services representative might check it for obvious errors. The following types of checks could be undertaken: *(1)* whether pages in a printed report are illegible because, say, a laser printer is low on toner; *(2)* whether the quality of film output is satisfactory; *(3)* whether tape cartridges or CD-ROMs have been labeled properly; *(4)* whether any pages in a printed report are missing; and *(5)* whether any pages in a report have been printed askew. This type of checking has two purposes. First, it provides a higher level of service by the information systems function to users. Users should obtain higher-quality output than they might otherwise receive. Second, if errors or irregularities in output are subsequently detected, it may be easier to detect their likely source. If pages in a report are missing, for example, responsibility can be assigned to user staff rather than to the information systems personnel who are responsible for production of the report.

4.3.7 Report Distribution Controls

If user/client services group representatives have responsibility for collecting output, they must ensure that it is distributed securely and promptly to the correct users. Distribution can occur in various ways:

1. Output might be placed in locked bins that users clear periodically.
2. Output might be delivered directly to users.
3. Output might be mailed to users, either via internal mail or via the normal postal service.
4. Output might be handed over to users or user representatives who present themselves to collect the output.
5. Output might be dispatched to users via a courier service.
6. Output might be given to a mail distribution organization; for example, customer invoices might be given to the organization to be stuffed in envelopes and dispatched.

To exercise control over output distribution activities, records should be kept of the date and time at which output was distributed and the identity of the person who received the output. Likewise, if output is given to a mail distribution organization, control totals should be kept and checked to ensure that all output is given to the organization and that the organization dispatches all output. These records provide a basis for identifying the nature and source of output errors or irregularities if they occur.

Controls must be in place to ensure that output is dispatched on a timely basis. If mailing of invoices is delayed, for example, an organization might experience cash flow difficulties. Similarly, managers could make wrong decisions if they do not promptly receive reports that notify them of

important changes in, say, their organization's financial position. Regular reviews should be undertaken, therefore, to ensure that output has been collected or distributed on a timely basis. These reviews should extend to third parties, such as mail distribution organizations, who distribute batch output.

Special care must be taken when a large number of copies of a report have to be distributed. A recipient name-and-address file should be maintained. To facilitate distribution of the report, the file then might be printed on gummed labels that are attached to individual copies of the report. If the user population changes frequently, the number of copies of the report required could vary from run to run. In these situations, maintaining the integrity of the name-and-address file is critical. If an unauthorized party were to insert their name and address on the file, a gummed label would be produced for them. They would then receive a copy of the report.

In some cases many different users collect output from a user/client services group. Alternatively, the collection is undertaken on behalf of users by a third party (e.g., a courier). Where users or third parties are unknown to the user/client services group, they should be asked to identify and authenticate themselves. Either prior authorization must have been given to hand over the output to the unknown user or third party. Alternatively, the users or third party should produce evidence to show they are authorized to collect the output.

4.3.8 User Output Controls

Users should perform reviews similar to those carried out by user/client services representatives in the information systems function to detect problems with output. Because users are more familiar with the application area, however, they are better placed to detect errors and irregularities in output. Moreover, periodically they should undertake an in-depth review of output to evaluate its quality. For example, they might perform test calculations to check the accuracy of extensions or control totals shown in an output report, or they might undertake a physical count of some inventory items to check whether the amounts on hand correspond to those shown in an inventory listing.

Procedures must be established to allow complete and timely reporting of output problems identified by users to someone who has overall responsibility for the quality of the application system that generated the output. Moreover, users must be properly informed of these procedures. If reporting procedures have not been established or users are unaware of these procedures, output problems that manifest important errors or irregularities in the application system might be overlooked or not identified promptly.

Controls should be in place to ensure that users review output on a timely basis. There is little point to providing users with prompt, accurate, and complete output if they then fail to attend to it within an appropriate time interval. Serious losses might arise as a result of users' tardiness in examining the output provided to them. For example, an organization might fail to adjust its portfolio of investments in light of important changes in the stock market. To ensure that output is used promptly, management might

undertake periodic reviews. In this light, employees might be required to endorse batch reports with the time and date of their review.

4.3.9 Storage Controls

Three major controls should exist in relation to storage of output. First, output should be stored in an environment that will allow it to be preserved for the period it is required. In this regard, various output media have different requirements in terms of the environments in which they should be kept. For example, although cartridges and CD-ROMs are fairly robust, nonetheless to maximize their lives they should be stored in environments that have a constant temperature and are dust free. Some laser-printed output will also deteriorate quickly if it is stored in hot, humid environments. Microfiche and microfilm should also be stored in cool, dry, dust-free environments.

Second, output must be stored securely. Stored output might contain confidential data that would lead to serious exposures if it fell into the hands of unauthorized parties. In some cases the stored output could be negotiable instruments, and an organization would incur direct financial losses if the instruments were stolen. Stored output might also contain data that is essential to the ongoing operations of an organization. If it were lost, damaged, or destroyed accidentally, the organization could suffer severe disruptions.

Third, appropriate inventory controls must be kept over stored output. For example, records must be kept of what output is in storage, where it is stored, and who has removed or returned output. To reduce the likelihood of loss or damage, the output should also be stored tidily in proper facilities. Periodically, checks should be made to ensure that the output that is actually stored matches the records of what is supposedly stored. If the output is kept by a third party, assurances must be obtained (perhaps from the third party's auditor) that the third party maintains adequate controls over storage.

4.3.10 Retention Controls

A decision must be made on how long each type of output will be retained. This decision can affect the choice of output medium and the way in which the output is stored. For example: reports produced by a laser printer deteriorate faster than reports produced by an impact printer; images stored on CD-ROMs deteriorate faster than those stored on microfiche; reports stored in warm, humid environments will deteriorate faster than those stored in cool, dry environments.

The decision on a retention period also affects how stored output must be managed. For example, output that has to be stored on magnetic media for long periods of time occasionally must be rewritten to ensure that it can still be read accurately and completely. Magnetic media will deteriorate because of stray magnetic fields, oxidation, or other types of material decay. Storage media also can become unreadable because they depend on old technology that becomes obsolete sooner than expected. For example, Rothenberg (1995) reports that the 1960 U.S. Census data was nearly lost because it was stored on magnetic tapes using formats that had been superseded by new, incompatible formats. Likewise, documents that were

created using word processors or spreadsheet packages at some time in the past say, ten years ago might no longer be readable because storage formats have changed. Recognition of such exposures requires careful management of stored output.

A retention date should be determined for each output item produced. The output item must then be kept until its retention date expires. Various factors affect the retention date assigned to an output item for example, the need for archival reference of a report, backup and recovery needs, taxation legislation specifying a minimum retention time for data, and privacy legislation specifying a maximum retention time for data.

4.3.11 Destruction Controls

When output is no longer needed, it should be destroyed. Report destruction can be accomplished easily using a paper shredder. As retention dates expire, reports should be transported in a secure manner to the shredding facility. Partially printed reports from aborted report runs and discarded stationery also should be shredded to prevent any unauthorized use.

4.4 Batch report design controls

An important element in the effective execution of production and distribution controls over batch output reports is the quality of their design. Good report design facilitates the orderly flow of reports through the various output phases examined in the previous section. Good management ensures that users employ these design features to comply with the control procedures laid down for batch reports during the output process.

Table 4-1 shows the information that should be included in a well-designed batch report to facilitate its flow through the output process and the execution of controls as it passes through each phase. The title page contains information that assists operators and user/client services personnel to perform their work. In an environment where large numbers of reports are produced, this information is especially important. If the same report is produced several times a day, or a report program has to be rerun for some reason, confusion can arise if each instance of a report is not identified uniquely.

The title page should also contain details about the person to contact in the event that operational problems have occurred in the production of the report; for example, the print might be. too light, or some pages might have been printed askew. This person can then *take* control over the defective report; for example, they might ensure that a confidential report is destroyed securely and authorize the production of another copy of the report. The title page should also contain details of the person to contact in the event that errors or irregularities are discovered on the basis of information contained within the report. Recall that we discussed previously the need for organizations to ensure that employees know the procedures they should follow when they discover errors and irregularities as a result of their examination of reports.

The information on the detail pages of a report prevents the unauthorized removal of data from the report. A person wishing to prevent a fraud from being discovered might remove a page containing exception information. Certain pages of a report might be especially valuable to a competitor. Page numbering and end-of-job markers are used, therefore, to prevent the unauthorized removal of a report page.

TABLE 4-1 Control information to be included in a well designed report

Control Information	*Position in Report*	*Purposes*
Report name	Title page	Permits immediate identification of report.
Time and date of production	Title page, detail pages	Prevents confusion when report is produced several times per day or when for some reason the report has to be produced again, e.g., an error in the report program.
Distribution list (including number of copies)	Title page	Facilitates distribution of the correct number of copies of the report to authorized users. Helps to ensure that unauthorized copies of the report are not produced.
Processing period covered	Title page	Users can see what data/time period has been covered by the report.
Program (including version number) producing the report	Title page	Permits immediate identification of originating system/program.
Contact persons	Title page	Indicates who should be contacted in the event that the report has been produced defectively and who should be contacted in the event that errors or irregularities are discovered on the basis of the report.
Security classification	Title page, detail pages	Alerts operators and user/client services representatives to the sensitivity of data contained in report.
Retention date	Title page	Indicates date before which the report should not be destroyed.
Method of destruction	Title page	Indicates whether special procedures need to be followed to dispose of the report.
Page heading	Detail pages	Shows content of report pages and perhaps security classification of data contained in report.
Page number	Detail pages	Prevents undetected removal of a report page.
End-of-job marker	Immediately after last entry, last page of report	Prevents undetected removal of last page of the report.

4.5 Audit trail controls

Audit trail controls in the output subsystem maintain the chronology of events that occur from the time the content of output is determined until the time users complete their disposal of output because it no longer should be retained.

Accounting Audit Trail

The accounting audit trail shows *what* output was presented to users, *who* received the output, *when* the output was received, and what *actions* were subsequently taken with the output. This information can be used for various purposes. For example, as discussed previously, it might be required to fulfill statutory obligations. Because of these obligations, an organization can be called to account for past events that have occurred. The accounting audit trail can provide the data needed to respond to any statutory request for information.

If an erroneous data item is discovered in an organization's output, the accounting audit trail also can be used to determine those users who might have relied on the output to make a decision. These users can then be notified to enable them to determine whether they need to take any actions to mitigate their exposure. If users who relied on the erroneous output are internal to the organization, it might be fairly straightforward to notify them promptly of the situation. If the erroneous output has been placed in a page on the Web, however, the situation is often problematic. The output might have been accessed by a large number of persons who are external to the organization. The organization must then determine what responsibilities it has, if any, to notify these users. Likewise, if an organization broadcasts erroneous output via a list server, it might be impossible to track down all the people who have relied on the output. The output might have been transmitted to other list servers and rebroad-cast multiple times. For this reason, organizations that make output publicly available often place a disclaimer with the output notifying people that they use the output at their own risk. Nevertheless, even if the disclaimer can be sustained in all cases of erroneous output, an organization might still want to notify users who have obtained erroneous output to reduce losses of goodwill that may arise.

The audit trail can also be used to determine whether unauthorized users have gained access to or unauthorized activities have occurred in the output subsystem. In this light, periodically management could examine the audit trail to determine whether the contents of output provided to users reflect improper access or improper activities. As a deterrent to irregularities, some organizations warn users that their activities are being logged. For example, an external user who gains access to an organization's Web pages via a browser might be informed that they should act appropriately and that all actions they take will be recorded.

Unfortunately, perhaps more than any other subsystem, efficiency considerations in the output subsystem dictate how complete the audit trail should be and how long it should be kept. Internal and external users of an organization's output subsystem can generate large amounts of batch and online output. The costs to store all this output for long periods of time can be excessive. Moreover, the exposures associated with much of this output

could be low. Accordingly, organizations must evaluate the exposures associated with different types of output. A decision can then be made on what output will be stored in the audit trail and the retention period that will apply to the different types of output.

Operations Audit Trail

Activities in the output subsystem can consume substantial resources; for example, expensive output devices might be required to produce graphs and images, consumption of costly, high-quality preprinted stationery could be high, large amounts of machine time might be required to produce image output, substantial storage space might be required to archive output, and a costly communications infrastructure might have to be implemented and maintained to support online access to output by both internal and external users.

The operations audit trail maintains a record of the resources consumed to produce the various types of output. For example, it can be used to record the number of report pages printed or the volume of output requested by external users. In addition, it might record data that enables print times, response times, and display rates for output to be determined. This data can then be analyzed to determine whether an organization should continue to provide different types of output to users. It can also provide information that enables the organization to improve the timeliness of output production and reduce the amount of resources consumed in producing output.

5. SUMMARY

The components in the input subsystem are responsible for bringing both data and instructions into an application system. Both types of input must be validated Moreover, any errors detected must be controlled so input resubmission is accurate, complete, unique, and timely.

The communication subsystem is responsible for transmitting data among all the other subsystems within a system or for transmitting data to or receiving data from another system. The integrity of data within the subsystem can be undermined by impairments in transmission media (attenuation, distortion, and noise), hardware and software component failure, passive subversive threats (release of message contents and traffic analysis), and active subversive threats (insertion, deletion, modification, and duplication of messages, changes to the order of messages, denial of messages services, and establishment of spurious associations).

The database subsystem provides functions to define, create, modify, delete, and read data in an information system. Several major types of controls must be implemented in the database subsystem to improve the reliability of its components and to protect the integrity of data stored in the database. Access controls restrict the actions that users can undertake on the database and the database definition to the authorized set of actions.

The output subsystem provides functions that determine the content of data that will be provided to users, the ways data will be formatted and

presented to users, and the ways data will be prepared for and routed to users.

6. QUESTIONS

1. From an audit perspective, why are controls over the input subsystem critical?
2. Briefly distinguish between direct entry of input data and medium based entry of input data. Why must auditors understand the different types of methods used to input data into an application system?
3. What techniques can be used to indicate the size of a field on a data-entry screen?
4. What attributes of a data code affect the likelihood of a recording error being made by a user of the code? Briefly outline some strategies to reduce error rates that occur with data codes.
5. Briefly describe the three major types of exposure in the communication subsystem.
6. What is meant by noise on a communication line? What factors affect the amount of noise that exists on a line? What are the effects of noise?
7. What control advantages do private communication lines offer over public communication lines?
8. What is meant by the topology of a network? List *three* factors that should be considered when choosing a network topology.
9. Distinguish between link encryption and end-to-end encryption. What are the relative strengths and limitations of link encryption versus end-to-end encryption?
10. What is communication architecture? How is the concept of communication architecture useful to us as auditors?
11. Briefly describe the functions of the database subsystem.
12. How can *views* be used to enforce access controls in the database subsystem?
13. Briefly describe the functions of the output subsystem.
14. Give *three* purposes to controls over printing of batch reports.

7. REFERENCE BOOKS

1. Weber R; Information Systems Control and Audit (Person Education)
2. Dube: Information systems for Auditing (TMH)
3. Auditing Information Systems, 2nd Edition. Jack J. Champlain (Wiley)

UNIT – IV

1. Audit software

Structure

Objectives

After going through this unit, you should be able to:

- understand how to Generalized audit software
- understand how to Industry-specific audit software
- discuss about High-level languages
- discuss about Utility software , Specialized audit software, and Control of audit software

1.1 Introduction

In this chapter we begin our discussion of various tools that auditors can use to collect evidence on the reliability of controls within an application system. Specifically, we focus on different types of software that auditors can employ to facilitate their evidence-collection work. A wide range of software now exists that auditors might find useful during an audit. In this light, we must have an understanding of the nature of this software, the functions it can perform, where it might be used within an audit, and its inherent strengths and limitations.

In the following discussion, we focus first on four types of off-the-shelf software that auditors often use during the evidence-collection phase of an audit: generalized audit software, industry-specific audit software, high-level languages, and utility software. We then examine two types of audit software that have their roots in artificial intelligence namely, expert systems and neural networks. Next, we discuss specialized software that auditors sometimes must develop and implement to address needs that cannot be met satisfactorily using off-the-shelf audit software. We then briefly examine some other types of audit software that have been developed over the years but have not enjoyed widespread use. By having an appreciation of this software, we can gain insights into how auditors use other audit software in innovative ways. Finally, we examine the approaches they can use to exercise control over audit software so that they have confidence in the integrity of the results they obtain using the software.

1.2 Generalized audit software

Generalized audit software is off-the-shelf software that provides a means to gain access to and manipulate data maintained on computer storage media. Auditors can obtain evidence directly on the quality of the records produced and maintained by application systems. In turn, their judgments on the quality of the records will enable them to make judgments about the quality of the application system that processes these records.

Generalized audit software packages first appeared in the mid-1960s. They were developed by several large public accounting firms to facilitate the audit work they needed to carry out on mainframe computers. Enhanced versions of these mainframe generalized audit software packages are still available and still used. Today, however, microcomputer-based generalized audit software packages are available. Data is often transferred from a mainframe to a microcomputer to enable auditors to work with a generalized audit software package. Over the years, the extent of usage of generalized audit software by auditors has been monitored. It has remained the most frequently used computer-assisted auditing tool.

1.2.1 Motivations for Generalized Audit Software Development

The primary motivation for developing generalized audit software is the set of problems caused by the diversity of computerized information processing environments that auditors might confront. The characteristics of information systems can vary considerably: different hardware and software environments, different data structures, different record formats, and different processing functions. With resource constraints, it is often impossible to develop specific programs for every system that will extract, manipulate, and report data required for audit purposes. Generalized audit software has been developed specifically to accommodate a wide variety of different hardware and software platforms. The trade-off is a loss of processing efficiency for the ability to develop quickly a program capable of accomplishing audit objectives in a new environment. In many cases, however, the loss in processing efficiency is more than compensated by savings in the labor hours required to develop audit software capabilities for specific computer systems.

A second major motivation for developing generalized audit software is the need to develop quickly an audit capability in light of changing audit objectives. Both external and internal auditors often face situations in which new audit objectives must be developed or existing audit objectives change. For example, the volume of transactions processed through an application system might increase markedly such that the system becomes a high-materiality system rather than a low-materiality system. Generalized audit software allows us to adapt quickly when these types of changes occur.

A third major motivation for developing generalized audit software is the need to provide audit capabilities to auditors who might be relatively unskilled in the use of computers. In the past, few auditors had extensive training and experience in computers. Today, many auditors have a broad, general understanding of computer systems. They might lack the specific knowledge and experience needed, however, to be able to cope with the different types of hardware and software platforms they confront. Most generalized audit software packages can be used by auditors who are not computer audit specialists. In this light, the computer audit capabilities of these auditors can be extended.

1.2.2 Functional Capabilities of Generalized Audit Software

Table 1-1 shows the major sets of functions that auditors can perform using generalized audit software. These functions can be executed using some type of high-level user interface, such as a graphical user interface.

TABLE 1-1 Major sets of functions performed by generalized audit software

Set of Functions	*Examples*
File access	Capabilities to read different data coding schemes, different record formats, and different file structures.
File reorganization	Sorting and merging files.
Selection	Boolean and relational operators available: e.g., A AND (B OR C); A EQ 2500; A GT 3000.
Statistical	Varies from sampling every *n*th item to support of attributes and variables sampling.
Arithmetic	Full set available: addition, subtraction, multiplication, division, exponentiation.
Stratification and frequency analysis	Capabilities to classify data into categories according to certain criteria.
File creation and updating	Capabilities to create and update work files based on an organization's production files.
Reporting	Editing and formatting of output.

In the following subsections, we examine each set of functions listed in Table 1-1 in more detail.

File Access

The file access functions enable files having different data coding schemes, record formats, and file structures to be read. Coding schemes like ASCII, EBCDIC, zoned, packed, and binary often can be read. Records can have fixed or variable formats. Typically, the file structures that can be accessed are sequential, index sequential, and random, although some packages provide access to more complex structures such as trees and networks. Some generalized audit software packages allow several files to be read simultaneously. Some also provide direct access to files created by several popular database management systems, accounting packages, spreadsheet packages, and word processing packages.

File Reorganization

The file reorganization functions allow data to be sorted into different orders and data from different files to be merged onto one file. Sorting capabilities are necessary for a variety of purposes for example, reporting data in a specified order or comparing data on two files. Merging capabilities are needed if data from separate files is to be combined on a separate work file. Functions can then be executed on this work file for example, statistical functions or various kinds of calculations.

Selection

Generalized audit software provides powerful selection capabilities for extracting data that satisfies certain tests. Typically, the Boolean operators AND, OR are provided as well as the relational operators EQ, GT, LT, NE, GE, LE that is, equal to, greater than, less than, not equal to, greater than or equal to, and less than or equal to. Brackets establish precedence. For example, the query (PAY GT 24000 AND (OVERTIME GE 6000 OR ALLOWANCES EQ 6)) would extract employee records in which pay is greater than $24,000 per year and overtime is greater than or equal to $6,000 per year, or employee records in which pay is greater than $24,000 and the allowances classification is category 6.

Statistical

The statistical capabilities of generalized audit software vary from moderately powerful to sophisticated. At a basic level, every nth record can be selected or records can be selected at random. Some packages also provide comprehensive attributes sampling, variables sampling, combined attributes/variables sampling, discovery sampling, stratified sampling, and dollar-unit sampling capabilities. Some also provide functions to support analytical review procedures for example, regression and financial ratio analysis capabilities. Selected data and key analytical review results from prior years can be saved and brought forward to facilitate analytical review in subsequent years. Generalized audit software also can be designed to provide input to separate statistical and financial modelling software where more powerful capabilities are required.

Arithmetic

Generalized audit software provides the full set of arithmetic operators enabling work fields to be computed, the arithmetic accuracy of data to be checked, control totals to be produced, and so on. For example, net pay calculations for a payroll file can be recomputed, or files can be crossfooted. Often the calculations can be made based on data from more than one input record. Calculated fields can be stored and then used in subsequent calculations. With some packages, calculated fields can also be used to develop tables for look-up purposes.

Stratification and Frequency Analysis

Generalized audit software packages often provide good capabilities with respect to stratification and frequency analysis. Different types of stratification, frequency analysis, and aging analysis can be undertaken. For example, the frequency of accounts receivable balances in certain classes can be determined: $0-$200, $200.01-$400, $400.01-$600, and so on. The distribution of accounts receivables balances could be an important determinant of the type of sampling method chosen or the substantive audit procedures chosen.

File Creation and Updating

Some generalized audit software packages allow work files to be created and updated. For example, the output could contain samples of input file records or user-defined records that include fields extracted from input records or calculated fields. In some cases, the output files can be written in formats that are suitable for input into widely used database management software, spreadsheet software, or word processing software. Auditors can use generalized audit software to extract the data needed for audit purposes from the application system files. By using the work file instead of the application system files, auditors then cause minimum interference to normal application system processing.

Reporting

Comprehensive reporting facilities are often available in generalized audit software packages. For example, free-form reports can be produced that allow auditors to control the title of the report, content of column headings, width of columns in the report, levels of subtotals, number of detail lines, page footers and page headers, and formatting of fields (such as zero suppression and the addition of dollar signs). Some reports that contain data that most auditors will require during an audit are produced automatically for example, reports containing control totals, record counts, negative amounts, and blank or zero fields. Special-purpose reports also may be available for example, those that will be sent to outside parties for confirmation purposes.

1.2.3 Audit Tasks that Can Be Accomplished Using Generalized Audit Software

Auditors can combine the functional capabilities of generalized audit software to accomplish several audit tasks:

1. Examine the quality of data,

2. Examine the quality of system processes,

3. Examine the existence of the entities the data purports to represent, and

4. Undertake analytical review.

In the following subsections, we briefly examine the ways in which these tasks can be accomplished.

Examine the Quality of Data

Auditors can use the functional capabilities of generalized audit software to examine the existence, accuracy, completeness, consistency, and timeliness .of data maintained on computer storage media. Consider the following examples:

1. Records for various fixed assets can be retrieved to see if, in fact, the records exist.

2. The calculation of sales discounts can be checked for accuracy.

3. The address field for customers in an accounts receivable file can be examined to see if it contains blanks.

4. Records on the personnel file and the payroll file can be compared for consistency.

5. A file of share prices can be checked to determine the last time it was updated.

Auditors might examine the quality of data maintained in application system files or in an organization's database for two reasons. First, the quality of the data reflects the quality of the application system that processes the data. For example, if the address field in a debtor's records is blank, we should question the adequacy of the validation processes contained in the system. Second, the quality of data reflects the quality of the personnel who developed and maintain the application system and the quality of the personnel who use the system. If the data is low in quality, the application system processing the data could be poorly designed, poorly implemented, or poorly maintained. If this situation has been allowed to continue, the quality of the personnel who use the system also must be questioned. Moreover, even if the system is well designed, implemented, and maintained, the data supplied by users might still be low in quality.

Examine the Quality of System Processes

Even though the quality of the data in an application system might be high, the quality of system processes still could be low from the viewpoint of achieving the objectives of an organization. For example, the data in an accounts receivable file might be accurate, complete, consistent, and timely. A substantial number of overdue accounts might exist, however, which reflects adversely on both the accounts receivable application system and the personnel who use the system. The system might not be producing adequate management reports to enable timely collection of receivables. Alternatively, the system might be producing adequate reports. Nonetheless, accounts receivable personnel may not be using the reports or not properly following up on collections.

Besides examining the quality of data, auditors can use generalized audit software to examine the quality of system processes in other ways. For example, in our accounts receivable example, auditors could use it to age the accounts receivable file to determine whether debtors were paying their accounts on a timely basis. Similarly, auditors could use generalized audit software to calculate inventory turnover statistics as a basis for identifying obsolete inventory. If they identified substantial amounts of obsolete inventory, auditors should then question the adequacy of system processes

for managing inventory. Moreover, auditors should also question whether inventory is overvalued.

Examine the Existence of the Entities the Data Purports to Represent

Data could exist and be accurate, complete, and consistent. It might not represent an object in the real world, however. For example, it might represent a bogus insurance policy or an inventory item that no longer exists. Auditors must determine, therefore, whether the entities that the data purports to describe really exist.

The statistical sampling capabilities of generalized audit software provide an important means of doing this. For example, auditors can use these capabilities to select a sample of debtors for confirmation or a sample of inventory for physical observation. The powerful reporting capabilities of generalized audit software can then be used to print confirmations in the form required for mailing to debtors or to sort and print inventory data in a way that will facilitate auditors' physical counts of inventory. They can then input the results obtained from their confirmations or physical inventory work to generalized audit software to obtain probabilistic statements about the number of errors or the size of the dollar error that is likely to exist in the accounts.

Undertake Analytical Review

Analytical review is the process of obtaining key ratios and totals from an organization's data for comparison with previous years' ratios and totals or industry wide ratios and totals. The information obtained from analytical review is used to support or question preliminary audit conclusions based on system reviews and other substantive tests. For example, a decline in the working capital ratio of an organization might be used to support a preliminary audit conclusion that the ongoing viability of the organization is at risk.

Auditors can use generalized audit software to support analytical review work in several ways. First, they can use generalized audit software to extract data required for analytical review from an organization's database or an outside database and to prepare various ratios and totals. Second, if generalized audit software provides regression analysis capabilities, auditors can use the software to examine firm and industry trends. Alternatively, they can use generalized audit software to prepare data in a format suitable for input to another package that provides regression capabilities or other kinds of modelling capabilities required. Third, auditors can use generalized audit software to maintain a database of key data and key indicators across time. For example, auditors might want to store key indicators from the current year's audit for future years so trends across years can be identified.

1.2.4 Functional Limitations of Generalized Audit Software

To use generalized audit software effectively and efficiently, auditors must understand both its capabilities and its limitations. The following subsections examine three limitations that undermine its usefulness as a means of collecting evidence on the reliability of information systems:

1. Generalized audit software permits auditors to undertake only *ex.post* auditing and not concurrent auditing.

2. Generalized audit software has only limited capabilities for verifying processing logic.

3. It is difficult for auditors to determine the application system's propensity for error using generalized audit software.

Ex Post Auditing Only

Generalized audit software enables evidence to be collected only on the state of an application system after the fact. In other words, the software examines the quality of data after it has been processed. Even if auditors use generalized audit software to undertake parallel simulation, the results produced by the parallel simulation program are checked against a set of existing results produced by the application system. Thus, some time lag will occur between an application system error occurring and its possible identification using generalized audit software. In some cases this elapsed time could be substantial if the application system is not audited on a regular basis.

For some types of systems, timely identification of errors could be critical. For example, consider a situation in which multiple online users access a shared database. Unless an error that occurs in a data item is discovered quickly, it could permeate the database and cause several incorrect decisions to be made by users. Timely identification and correction of the error is therefore critical. In this light, Chapter 18 discusses the use of concurrent auditing techniques that permit evidence to be collected, and sometimes evaluated, at the same time as application system processing occurs. Auditors must use specialized rather than generalized audit software, however, to implement concurrent auditing techniques.

Limited Ability to Verify Processing Logic

Often the tests performed with generalized audit software involve "live" data that is, data captured and processed by the application system during the normal course of business. The limitations of using live data to test application systems, however, are well known. The data might not manifest the exceptional conditions that occur occasionally within the application. As a result, the application system's capability to handle these exceptional conditions accurately and completely is not tested. To overcome this problem, test data must be de signed specifically to determine how the

application system handles exceptional conditions. (Chapter 17 discusses this issue in more detail.)

Limited Ability to Determine Propensity/or Error

Systems can be designed and implemented in ways that allow them, at least to some extent, to accommodate change. For example, database management systems can be used to isolate certain types of changes to the database design from the application systems that access the database. Alternatively, application systems can be designed and implemented in ways that cause them to degenerate quickly when change occurs. Perhaps they are written in this way to allow them to process data efficiently.

1.2.5 Accessing Data with Generalized Audit Software

The way auditors access data with generalized audit software will depend on whether the data they need to access resides on the same machine as the generalized audit software package they are using. If the data they need to access resides on the same machine as their package for example, they are using a mainframe-based generalized audit software package access usually is straightforward. They simply define the file, record, and data formats of the file they wish to access. Their generalized audit software package then should be able to access the file. In some situations, the file might be stored using a format that cannot be read by their generalized audit software package. They then might have to use some type of utility to convert the file into a format that can be read by their generalized audit software package.

To illustrate how auditors might undertake a file transfer, assume that the file resides on a mainframe and the generalized audit software package resides on a microcomputer. Several methods can be used to undertake the file transfer:

1. If the file to be transferred is large, it can be written to a cartridge or perhaps a tape. Before transfer, it might be useful to sort the file in the order that will be most useful to the generalized audit software application. It might also be useful to select only certain fields in records for transfer or certain records for transfer. These actions will reduce processing time on the microcomputer. The microcomputer on which the generalized audit software package resides must have a device that can read the cartridge or tape.

2. If the file to be transferred is small, it can be written to a diskette. The diskette can then be transported to the microcomputer on which the generalized audit software resides.

3. A modem plus some type of file transfer utility can be used to transfer the file from the mainframe computer to the microcomputer.

4. The mainframe can be connected via a gateway to a local area network. The microcomputer on which the generalized audit software resides is also connected to this network. A file transfer utility can then be used to transfer the file to be accessed from the mainframe to the microcomputer.

1.2.6 Managing a Generalized Audit Software Application

Whenever auditors develop a generalized audit software application will be simple and require few resources. Accordingly, project management concerns will be minimal. In other cases, however, the application will be complex and costly. Careful control will have to be exercised, therefore, to ensure that the development work is carried out effectively and efficiently. In the following subsections, we briefly examine the work that should be undertaken during the major phases of developing and implementing a generalized audit software application.

Feasibility Analysis and Planning

Certain types of generalized audit software applications are costly to develop, implement, and operate. Auditors should take care to estimate the costs and benefits of a generalized audit software application whenever it has some or all of the following characteristics:

1. The application will access large data files such that the execution time required will be substantial.

2. The application is complex such that the development and testing time required will be substantial.

3. The application system to be evaluated using generalized audit software is poorly documented and poorly understood.

4. The application system to be evaluated using generalized audit software is subject to frequent changes that will require the generalized audit software application to be modified frequently.

5. The application is to be run frequently; it is not a one-off or occasional use o generalized audit software.

6. The audit staff are not skilled users of computer systems, or they lack knowledge of and experience with the generalized audit software package.

7. The audit objectives to be accomplished with the generalized audit software application are unclear or still evolving.

When auditors estimate the costs associated with a generalized audit software application, they must ensure that they take into account all the material costs that are likely to be incurred. Many different types of costs can arise for example, labor costs associated with understanding, developing, implementing and operating the application; labor costs

associated with application systems staff and technical and administrative staff who must provide assistance; computing costs associated with the development, testing, and execution of the application; supply costs such as those associated with the special stationery used for confirmation letters; and costs of disruption if errors are introduced into the application system as a result of their using generalized audit software. Similarly, many different types of benefits can accrue from using generalized audit software for example, reductions in the amount of time to perform the audit, improved detection of errors and irregularities, and improved feedback to the users of application systems on the quality of controls. Again, auditors must be sure that they take all material benefits into account when determining whether a generalized audit software application will be worthwhile.

Having estimated the costs and benefits associated with a generalized audit software application, auditors should then prepare a budget to determine whether the benefits of the application are likely to exceed the costs. The budget will also form the basis for controlling the development, implementation, and operation of the application. Auditors also need to prepare a timetable for development to ensure that the application can be developed within the required timeframe.

Application Design

During the application design phase, auditors undertake detailed design of the generalized audit software application. The amount of work done during this phase will depend on such factors as the size, complexity, and criticality of the application. For example, if the application is large and will consume substantial amounts of machine time to execute, auditors should take substantial care to ensure that the logic is correct from the outset and that the generalized ''audit software package will perform the functions required as efficiently as possible. On the other hand, if the application is small and requires few machine resources, auditors might use a prototyping approach when preparing the application. For example, auditors might prepare a quick, preliminary design with a view to obtaining a working version of the application that they can use on an experimental basis.

Some of the important design steps to be undertaken during this phase are as follows:

Design Step	*Explanation*
Obtain detailed understanding of application system to be audited	Auditors can obtain this understanding by reviewing application system documentation, flowcharting the application, preparing decision tables, and so on.
Design output reports required	The output reports constitute some of the working papers for the audit. Auditors must ensure that the information they require for the audit is output in a convenient form.
Prepare file definitions	Auditors must define any files to be accessed and any work files to be created.
Define the logic of the audit software program	Auditors must determine what functions they want the generalized audit software application to perform. The logic of the application should be documented in some way—e.g., flowcharts or decision tables.
Define any supplementary data needed	Additional data, such as reference (look-up) tables, may need to be defined.
Prepare a test plan	Testing can be undertaken using a test deck or live data. Chapter 17 discusses various techniques for testing programs.
Desk check logic	When the logic of the generalized audit software application has been defined, auditors should undertake desk-checking to determine whether their audit objectives will be met.

When auditors have completed the design phase, they might need to review and to evaluate the design against the budget and timetable prepared during the feasibility analysis phase. In light of the design prepared, it might be clear that they cannot achieve the outcomes required of the generalized audit software application within the budget or timetable prepared. More resources or additional time might have to be allocated to the application. Alternatively, the application might have to be scrapped or modified in some way.

Coding and Testing

After the design has been completed, the application must be coded and tested. In some cases, auditors simply input commands to the generalized audit software package and observe the outcome as each command is interpreted and executed. They are likely to follow this approach if the results from each command can be obtained quickly. In other cases, however, auditors might write a series of commands and store them in a command file that is subsequently read and executed by the generalized audit software package. They are likely to follow this approach if the commands take substantial time to execute. The command file might be read and invoked at some time during the night when the load on the

machine used to execute the generalized audit software application is low. Because online and batch execution of commands is often needed, some packages allow a command file to be created for subsequent use as auditors input each command and observe its outcome. Thus, auditors can develop the application iteratively and store the final version for subsequent execution as a command file.

Auditors should ensure that the results produced by a generalized audit software application are correct and complete. For example, they might select a few input records from the application system's files and validate the output produced by the generalized audit software program in light of these input records. In this regard, some packages provide a facility to select a small number of records from a file to enable testing of the generalized audit software application to be undertaken before the entire file is accessed. If the generalized audit software program will consume substantial resources during its execution, testing the application becomes especially critical before production running is undertaken.

Operation, Evaluation, and Documentation of Results

As discussed previously, sometimes operation of a generalized audit software application simply involves submitting input commands to the package that are interpreted and executed immediately. If auditors prepare batch command files for delayed execution, however, they might need to establish a processing schedule with the operations manager. Furthermore, if some time has elapsed since the audit software application was last run, auditors should check to see that no changes have been made to the application system that would impact its completeness and correctness. For example, a change to a record format could cause the application to produce erroneous output.

Upon obtaining the reports produced by the audit software application, auditors should review the output to check for any errors, derive a set of audit conclusions, and determine whether audit objectives have been attained. Re-specification and rerunning of the application might be necessary. The output must be incorporated into the audit working papers along with audit conclusions and any suggestions for improvements in future use of the application. In this regard, some packages prepare output and an audit trail to facilitate preparation of audit working papers. The costs and benefits of the application also should be compared with the budget. Finally, any files created by the application that might be needed for future use should be secured.

1.3 Industry-specific audit software

Some types of audit software packages are now available that are oriented toward specific industries for example, the financial services, health care, and insurance industries. The packages are still *generalized* because they provide auditors with high-level languages that can be used to invoke a wide range of functions. They differ from the types of audit software

examined previously, however, in two ways. First, because they are oriented toward a particular industry, they provide high-level commands that invoke common audit functions needed within the industry. For example, in the banking industry, they might use a single command to invoke logic that would check for account kiting. If generalized audit software were to be used to check for kiting, several commands might be required to express the logic needed to execute the various tests. Second, industry-specific audit software could have been developed to access data maintained by a specific generalized application package that is used extensively within the industry. Accordingly, the file, record, and field definitions used by the application package could be built into the audit software package; that is, auditors do not have to provide these definitions each time they want to run the package.

The CAPS package developed by Brisbane-based Kendalls Chartered Accountants is an example of an industry-specific audit software package. It has been designed for auditors of financial institutions (primarily auditors of credit unions and building societies). As such, it provides high-level commands to invoke functions that they will need. In addition, CAPS has been written to access the data maintained by two widely used generalized application packages within the finance industry. Indeed, CAPS cannot be used unless auditees employ one of these two packages for their basic application processing. If the auditee uses one of those packages, however, CAPS provides nine major sets of audit capabilities:

1. *Loan arrears audit.* CAPS can be used to evaluate the movements in loan arrears on a member's loan balance throughout a specified period. For example, a report is provided showing any case in which a new disbursement has occurred in spite of the loan being in arrears. Using this type of information, auditors could assess the auditee's controls over loan arrears.

2. *Interest audit.* This module recalculates all interest on member loans and savings accounts to provide an independent check on calculations carried out by the application system.

3. *Term deposit interest audit.* This module recalculates total term deposit interest to a specified date to provide an independent check on calculations carried out by the application system.

4. *Member ledger balances audit.* This module provides several functions that assist auditors to evaluate the veracity of member ledger balances. For example, it provides summarized information on each loans, savings, and investments ledger; it allows stratified sampling of member ledger balances for confirmation purposes; and it provides routines to statistically evaluate the results of a confirmation of members.

5. *Member ledger transactions audit.* This module examines ledger transactions for evidence of unusual circumstances. For example, it identifies transaction values outside a specified range; it identifies when

a disproportionate number of a particular transaction type has occurred; and it selects transactions randomly for audit scrutiny.

6. *Member biographical audit.* This module examines the reasonableness of various demographic and personal data held about a member. For example, it looks for member names without vowels (unusual names); it tests for post codes (zip codes) outside a particular range; and it tests for members who have a post office box number as a primary address.

7. *Dormancy audit.* This module identifies member accounts that are dormant and that, as a consequence, bear a greater risk of fraudulent or unauthorized transactions remaining undetected for some time. The module retains a separate file of dormant accounts and provides a report on changes to the file when subsequent dormancy audits are conducted.

8. *Incompatible duties audit.* This module allows the set of transactions that different operators (tellers) are allowed to execute to be defined. It will then check transaction log files to determine whether any operators are executing transactions that manifest inadequate separation of duties.

9. *Legislative compliance audit.* This module determines whether the financial position of an organization complies with legislative requirements. For example, a credit union might have to ensure that the proportions of its loans maturing within 3 months, 6 months, 9 months, 12 months, and greater than 12 months fall within certain ranges. Otherwise, it will be in breach of legislation.

The primary advantages of industry-specific audit software over generalized audit software are that it runs more efficiently and that it is easier to use because it incorporates higher-level functions. The primary disadvantage is that. it has a more limited domain of application than generalized audit software. As such, it tends to be more useful for internal auditors or external auditors who perform a large number of audits within a specific industry.

1.4 High-level languages

Besides generalized audit software, auditors can often use a high-level language to gain access to data and manipulate this data. In particular, many auditors now use fourth-generation programming languages, such as SQL and QBE, and generalized statistical software, such as SPSS™ and SAS®, to collect evidence on system reliability.

Fourth-generation languages have proved useful to auditors' work for several reasons. First, most functions incorporated within generalized audit software packages are also included within fourth-generation languages. For example, auditors can use fourth-generation languages to select data from files that satisfy certain criteria and to format this data for reporting purposes. They might have weaker capabilities in a few areas for example, statistical sampling capabilities. Often auditors can overcome these difficulties, however, by using "macros," which allow them to write programs

(perhaps using the high-level language) to perform particular functions and then to invoke these programs with a single command. Furthermore, some vendors of fourth-generation languages have adapted their software to produce specialized versions for auditors that contain, for example, statistical sampling functions.

Second, for the types of functions auditors might want to perform, fourth-generation languages could be more user friendly than generalized audit software. For example, a fourth-generation language might provide them with more flexible reporting capabilities. Auditors might also be able to avoid difficult downloading of data from one computer to another computer or troublesome conversion of one file or data format to another file or data format.

Third, if auditors use a fourth-generation language that is employed extensively throughout the organization audited, they are likely to be able to get good support to overcome any difficulties they might encounter. For example, if the organization uses a relational database and SQL, many persons within the organization should be able to assist auditors if they have problems using SQL to access and manipulate data in the database.

Many auditors have also become more frequent users of statistical packages because they now place increased reliance on analytical review as a diagnostic tool in the conduct of audits. In some generalized audit software packages, the statistical capabilities provided are fairly basic. They are oriented primarily toward support of statistical sampling activities. Analytical review often relies on using other statistical models, however, some of which are complex and require substantial computational support. For example, if auditors undertake time series modelling, they need various types of linear and nonlinear regression models; if auditors develop bankruptcy prediction models, they need discriminant analysis models. Statistical packages often offer very powerful modelling capabilities, and these capabilities are continually being enhanced. Moreover, the user interfaces are friendly, and highquality help functions and documentation exist to support their users.

As with generalized audit software, the widespread deployment of microcomputers has contributed significantly to auditors' increased use of fourth-generation languages and generalized statistical packages. With suitable utility software, auditors can download or transfer a copy of the data they need from another computer. The microcomputer versions of fourth-generation languages and statistical packages can then be employed to access and manipulate the data and prepare reports. In this way, auditors can work in a standardized environment rather than having to deal with multiple hardware/software platforms.

1.5 Utility software

Utility software is software that performs fairly specific functions that are needed frequently, often by a large number of users, during the operation

of computer systems. For example, they include copy programs, sort programs, disk search programs, and disk formatting programs. They often come as part of the suite of programs provided with major system software, such as operating systems, database management systems, fourth-generation languages, or data communications software. Much independent utility software has now been developed, however. It can be purchased to undertake functions that cannot be accomplished using the utility programs provided with system software or alternatively to undertake functions more effectively and efficiently than the utility programs provided with system software. Some also exists as freeware or shareware (although free use might be restricted only to personal use rather than business use). It might be downloaded, for example, from a site on the Internet. Auditors use utility software *tor five* reasons:

1. Utility software might have been developed to perform a specific security or integrity-related function. For example, auditors might use a utility program to check for viruses on a disk.

2. Before auditors can use generalized audit software or other types of audit software, they might need to format and download data using utility software.

3. Utility software might perform functions that cannot be performed using generalized audit software or other audit software available. For example, auditors might use a utility program to try to recover a damaged disk file that contains data that is material to the audit. It is unlikely that audit software will be able to perform this function.

4. Utility software might accomplish audit tasks more effectively and more efficiently than audit software. For example, it might be possible to select certain kinds of data and print a report using generalized audit software. Utility software might perform the same functions but consume fewer resources and prepare better-formatted reports.

5. Auditors might use utility software to assist with the development of new audit software. For example, they might seek to develop audit modules that they can embed in application systems to collect evidence at the same time that application system processing occurs. Auditors might use utility software to help test whether the modules work accurately and completely before they release the modules into production.

The following utilities illustrate those auditors might find useful from time to time:

Utility	*Function*
Virus scanner	Checks to determine whether a virus has corrupted a disk (Figure 16–7).
Damaged disk recovery	Diagnoses and attempts to repair damaged disks. Disk can be recovered in part or in total.
Unerase	Searches for and recovers erased files. Can be useful if data has been deleted to hide an irregularity.
Undo format	Recovers data if it has been lost because disk has been reformatted.
Software inventory manager	Identifies illegal software that is being used on microcomputers or local area networks. Can prevent installation of illegal software. Reduces exposures from software copyright and licensing violations. See Stocks et al. (1997).
Static security analyzer	Analyzes the security-relevant parts of a computer's file system. Detects whether the computer's security. state and configuration files have changed in undesirable ways.
Dynamic security analyzer	Detects anomalous behavior on the part of users or patterns of events that are known to reflect misuse of a system.
Dial-up access risk analyzer	Detects unauthorized dial-up nodes in a network.
Access control analyzer	Analyzes how well access controls have been implemented and maintained within an access control package.
Invalid Social Security Numbers	Detects invalid U.S. Social Security Numbers.

1.6 Expert systems

Expert systems are programs that encapsulate the knowledge that human experts have acquired about a particular domain and possess capabilities to reproduce this knowledge when presented with a particular problem. Throughout the 1980s, several audit firms, internal audit groups, and independent vendors expended substantial resources to develop expert systems to assist with audit work. In light of the success enjoyed with these expert systems, development work has continued. As a result, auditors can now use expert systems to assist them with both evidence-collection and evidence-evaluation activities.

Motivations for Using Expert Systems

There are three major reasons why auditors might develop, maintain, and use expert systems:

1. Expert systems make available to many auditors the knowledge typically possessed by only a few auditors. By definition, expertise is a scarce resource. When expertise is embodied in an expert system, however, it

can be accessed and used widely without the expert having to be present. Thus, expert systems provide a mechanism for effectively disseminating and operationalizing expertise in the audit domain.

2. Because computer technology evolves rapidly, it is difficult for auditors to remain knowledgeable across the range of technologies they are likely to confront in an audit. They might attempt to handle this complexity by designating certain audit colleagues as having responsibility for remaining current in a particular technology, embodying their expertise in an expert system, and disseminating their expertise via the expert system.

3. Expert systems provide a mechanism for increasing consensus and consistency in auditors' evaluation judgments. Because expert systems can be used to guide auditors through a series of judgmental steps, they help ensure that (a) important judgments are not omitted; (b) auditors are aware of significant information that may affect their judgment; (c) auditors are alerted to judgmental inconsistencies; (d) auditors are aware of alternative judgments that might be made on the basis of the evidence available; and (e) auditors maintain a proper record of documentation to support their decision making.

1.7 Specialized audit software

Specialized audit software is software written in a procedure- or problem-oriented language to fulfill a specific set of audit tasks. The term "specialized" does not mean the software performs only a narrow range of functions. Indeed, in some cases the software has extensive functionality. Rather, specialized means auditors have developed and implemented the software where the purposes and users of the software are well-defined before the software is written. On the other hand, with generalized software, the specific tasks to be undertaken by the software and the identity of users will not be known at the outset.

1.7.1 Reasons for Developing Specialized Audit Software

There are six reasons auditors might develop specialized audit software:

1. *Unavailability of alternative software.* Occasionally, auditors might encounter situations in which no generalized software is available to perform audit procedures. For example, the auditee might have developed or purchased some type of specialized hardware platform on which only a minimal suite of software will run.

2. *Functional limitations of alternative software.* Even if auditors have generalized software available to perform an audit task, its functionality might be limited. For example, government auditors sometimes undertake complex information processing activities to check for errors and irregularities. They match data from tax returns, bank accounts, share transactions, welfare payments, and so on, to identify whether citizens are defrauding their government. The generalized software

available to government auditors might not be capable of processing the large number of files that must be matched concurrently nor handling the complex data formats and file structures that have been used.

3. *Efficiency considerations.* In some cases, the audit tasks to be undertaken consume substantial resources, perhaps because auditors have to access large databases or have to perform audit tasks frequently. For example, in the complex matching task sometimes undertaken by government auditors examined previously, processing efficiency is often a primary objective. The matching task can be very costly because large, complex data files have to be processed. In this light, government auditors often develop specialized audit sdftware because it will perform the matching task more efficiently than generalized software.

4. *Increased understanding of systems.* Sometimes the systems to be audited are complex. Nonetheless, it is important that auditors gain a proper understanding of the system as a basis for conducting the audit. One way that they might seek to gain this understanding is to prepare program specifications and to write the source code for specialized audit software. In the case of the computer matching example examined previously, government a valuable insights into the application systems that process the files used in the matching task if they participate in the development and implementation of the specialized software used to carry out the matching task.

5. *Opportunity for easy implementation.* Opportunities sometimes exist to develop and implement specialized audit software quickly and easily. For example, auditors might be able to insert a few instructions in an application system that gathers data that is critical to a judgment on the reliability of controls in an application system.

6. *Increased auditor independence/respect.* To the extent that auditors develop their own software and are not reliant on the auditee to provide software or staff support, they are more independent in the conduct of their audit. Moreover, auditors have an opportunity to demonstrate professional competence to the auditee. As a result, the auditee might have increased respect for their work. In the case of the government matching example examined previously, this respect and confidence in the auditor's abilities could be essential to obtaining support by the legislature to continue matching work. If the legislature loses confidence in its auditors, it might deem that the rights of individual persons overrule the need to search out fraud. As a result, it might order that the matching activities be stopped.

One important area where auditors often have to prepare specialized audit software is in the development and implementation of concurrent auditing techniques. Concurrent auditing techniques collect audit evidence at the same time as the application system is processing production data. They require audit hooks, modules, or routines to be embedded in the application

system to select the evidence required. These are often implemented via specialized program code.

1.7.2 Development and Implementation of Specialized Audit Software

Specialized audit software can be developed and implemented in three ways. First, auditors can take total responsibility for developing and implementing the software themselves. This approach allows auditors to exercise a high level of control over the software. To produce high-quality software, however, auditors must possess good analysis, design, and programming skills. Second, internal auditors can ask programmers in their own organization to develop and implement the software. Alternatively, external auditors can ask programmers in the client organization to develop and implement the software. Third, auditors could ask an outside software vendor to prepare the software. Auditors might adopt this approach if the software is especially sensitive. Though the costs might be higher, using the services of an independent third party provides extra assurance that integrity violations have not occurred during the development and implementation process.

Whatever approach auditors use to develop and implement specialized audit software, they must exercise careful control over the development and implementation process to ensure that the software meets their objectives and the integrity of the software is preserved. Auditors can exercise most control when they prepare the software themselves. If auditors use other personnel to prepare the software, however, they should still take responsibility for preparing program specifications, managing the programming process, performing acceptance testing, and preparing user documentation. Unless auditors perform these tasks, they must be circumspect about placing reliance on the integrity of the program.

1.8 Control of audit software

When using audit software for evidence-collection purposes, auditors should evaluate the level of control they are able to exercise over the software. If auditors have to employ software controlled by other parties, they run the risk that the software might have been modified improperly (either deliberately or unintentionally). The results produced using the software, therefore, might not be accurate or complete. Similarly, if auditors must rely on other parties to execute software on their behalf, they run the risk that the results obtained will lack integrity. The execution of the software might have been compromised in some way. Alternatively, a mistake might have been made when executing the software.

If auditors have to employ software controlled by other parties, there are two ways they can determine whether the software has been modified:

1. *Hash total.* At a prior time, auditors might have been able to obtain or calculate a hash total of the object code for the software. They can then calculate the hash total for the object code of the program provided and

compare the result obtained with the previous hash total calculated. If auditors are to be able to rely on the results, they need to have control over the program that computes the hash total.

2. *Test data.* Auditors can develop test data to test out those functions in the software on which they intend to rely. Unless auditors are confident that the software has not been modified since they last used it, they will need to execute the test data each time they employ the software for evidence collection purposes.

If auditors rely on other parties to execute software on their behalf, they must carefully examine the results provided for evidence of any errors or irregularities. Auditors might desk check a sample of computations performed by the software, for example, to satisfy them selves that the software has been executed properly.

If auditors maintain an independently controlled library of audit software, unauthorized modifications to the software are less likely to occur. Auditors can protect the library via access controls. Moreover, they might be able to maintain the software on a machine that they control. For external auditors, maintaining an independent library on their own machine has an additional advantage: Providing they can download data to their own machine, they are less constrained in carrying out audit work by the availability of software on the client platform audited.

2. Code review, Test data, and Code comparison

Structure

Objectives

After going through this lesson, you should be able to:

- understand where do program defects occur?
- understand how to Test data
- understand how to Program code comparison

2.1 Introduction

In this lesson, we examine three evidence-collection techniques used during substantive testing to determine primarily whether *(1)* programs meet their functional requirements and *(2)* program code is defective because it is unauthorized, inaccurate, incomplete, ineffective, or inefficient. The first technique is *program code review,* whereby we obtain program source-code listings and read these listings to evaluate the quality of the program code. The second technique is *test data,* whereby we design a sample of data to be executed by the program and then examine the output produced to reach a judgment on the quality of the program. The third technique is *code comparison,* whereby we compare two versions of a program's source or object code. One version the blueprint has known attributes, and we seek to determine whether the other version has the same attributes.

Each of these techniques can be used independently. Sometimes, however, we might choose to use them in a coordinated way (Figure 2-1). First, we undertake program code review to generate hypotheses about deficiencies in the program code for example, errors or inefficiencies. Next we design and execute test data to test these hypotheses. We can then ask that any deficiencies we confirm using the test data be corrected. These two steps might be carried out iteratively until we are satisfied with the quality of the program code. The resulting version of the program then becomes a blueprint. At a later time, we then compare production versions of the program against this blueprint to determine whether any discrepancies exist.

In the subsequent discussion, we first examine where we are likely to identify defects in a program. By knowing this we can better focus our efforts during program code review and test-data design and execution. We then discuss the nature of program code review, test data, and code comparison, their use in our evidence-collection work, and their strengths and weaknesses.

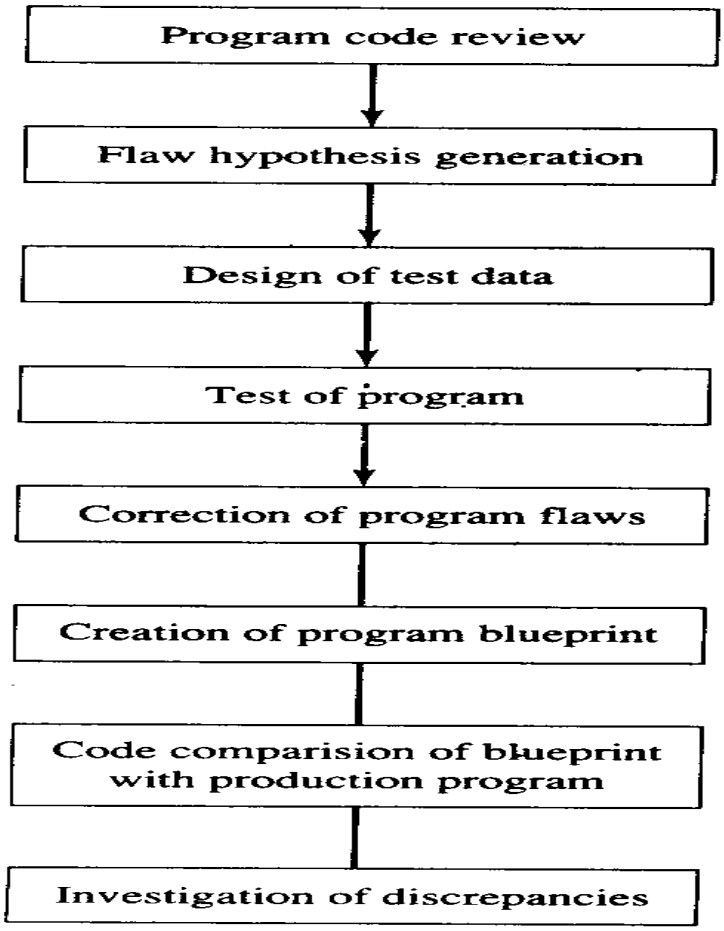


FIGURE 2-1 Integrated use of code review, test data, and code comparison for evidence-collection

2.2 Where do program defects occur?

Code review and test data can be time-consuming evidence-collection techniques to use. Auditors should apply them, therefore, where they will have most effect. In this light, it would be useful if we knew those program locations where defective code is most likely to occur.

In spite of the enormity of the programming effort worldwide, unfortunately, few rigorous studies of where the problem areas in programs lie have been reported. The bibliography at the end of this chapter contains references for several of these studies. Here are a few tentative conclusions, however, that we can draw based on the studies conducted so far:

1. Pareto's law seems to apply in programming: A small number of program modules will have a large number of faults, and a large number of modules will have none or only a few faults. Moreover, a

small number of faults affect a large number of modules, and a large number of faults affect only a small number of modules.

2. Requirements specification and design errors are just as prominent, if not more prominent, than coding errors. Auditors should be careful, therefore, if they rely on the accuracy and completeness of the program requirements or program design in their evidence-collection work.

3. Design errors seem to relate primarily to interface problems with users, input/output devices, and the database. Many design errors also reflect that the programmer has failed to implement part of the specifications—in other words, logic has been omitted.

4. Coding errors seem to relate primarily to incorrect computation, incorrect indexing, or incorrect control flow.

5. Defects relating to management of data seem to be more common than defects relating to computation.

The extent to which defects exist in a program is also likely to depend on the age of the program. The more times a program has been used for production purposes, the more likely it is that defects will have been identified and corrected. Thus auditors often should he able to limit the extent of the testing undertaken with older programs. In some cases, however, organizations have learned to live with defective older programs, in which case auditors might need to expand the extent of their testing.

2.3 Program source-code review

Auditors will use program source-code review when they are unwilling to treat a program as a black box. In other words, they decide that they are not prepared to make inferences about the quality of a program's code based only on an examination of the program's input and output. Instead, they wish to look at the internal workings of the program to evaluate its quality.

2.3.1 Objectives of Code Review

When undertaking code review, it is important that auditors have the objectives they wish to accomplish clearly fixed in their minds. The code review task is complex, and they can be distracted easily if they do not have clear goals. The objectives auditors might seek to accomplish are the following:

1. *Identify erroneous code.* The use of code review to identify erroneous code is well established. Recall that we discuss this purpose in Chapter 5 under various headings: desk checking, structured walkthroughs, and design and code inspections. The empirical evidence we examined earlier in the chapter indicates that coding errors are still a major cause of low-quality programs. Thus, auditors can use code review to determine whether program code complies with its specifications.

2. *Identify unauthorized code.* Without directly examining a program's source code, auditors are unlikely to identify unauthorized code in a program. Unauthorized code often is triggered by a specific data value or combination of data values. For example, a fraudulent programmer might modify a program so it does not print out details of his or her own account when it is overdrawn. Similarly, he or she might modify a program to exclude transactions having certain account number and date values from normal data validation processes. Unless auditors submit test data having these specific values and have a way of checking that the test data has traversed all execution paths in the code that is their focus, they are unlikely to detect this unauthorized code.

3. *Identify ineffective code.* Auditors can examine whether code is ineffective in two ways. First, they can evaluate whether the code meets the documented program specifications. Second, they can examine whether the code meets user requirements. Recall, the empirical evidence discussed earlier in the chapter suggests that documented program specifications and user requirements often do not correspond. Moreover, assuming the program specifications are correct, design errors that result in the program not complying with specifications are' also prevalent.

4. *Identify inefficient code.* Code review also can allow auditors to identify inefficient segments of code. For example, in a sequence of tests of transaction types, the tests might not have been ordered according to their frequency of occurrence. As a result, the program executes more of its code than it would have to if the tests were reordered. Auditors might also use code review to identify the existence of instructions that execute inefficiently on the hardware/software platform used.

5. *Identify nonstandard code.* Nonstandard code takes a variety of forms. For example, it could be code that does not comply with organizational standards covering data item names or internal documentation. Alternatively, it could be code that does not employ structured programming control structures. Whatever the nature of the nonstandard code, often it manifests other defects in the code for example, unauthorized code or erroneous code.

Which of these objectives should be emphasized within code review will depend on the objectives auditors have for the audit and the nature of the material exposures they conclude exist. For example, if their primary concern in the audit is the integrity of the data in the financial statements, their focus will be on whether unauthorized or erroneous code exists. If auditors conclude, however, that there are few opportunities for defalcations associated with the code they are examining, they will probably narrow their focus to determining whether erroneous code exists.

2.3.2 Source-Code Review Methodology

A review of program source code involves seven steps:

1. Select the source code to be examined.
2. Review the organization's programming standards.
3. Obtain an understanding of the program specifications.
4. Obtain the source-code listing.
5. Review the programming language used to implement the code.
6. Review the source code.
7. Formulate flaw hypotheses.

The following subsections briefly discuss each of these steps.

Source-Code Selection

Program source-code review can be a time-consuming evidence-collection technique to use. In this light, auditors should select source code for review that is material to their audit objectives. Risk assessment techniques can be applied to determine the level of materiality associated with the source code. For example, Sherer and Paul describe a methodology for identifying high-risk program modules. It involves five major steps:

1. The hazards (e.g., errors or irregularities) that can occur with a program are identified.
2. The expected losses associated with the occurrence of each hazard are estimated.
3. The exposure associated with each program module is estimated by relating factors like the module's functions and frequency of use to the hazards and their financial consequences.
4. A software reliability model is used to determine the expected number of failures that will occur in each module as a result of software faults.
5. The expected losses of failure in each module are estimated based on the likelihood of it failing, the hazards that will arise as a consequence, and the expected losses associated with these hazards.

Review Programming Standards

By reviewing the organization's programming standards, auditors develop a set of expectations about the characteristics of the code they will review for example, the way labels will be assigned to variables and constants, the way programs will be structured, and the way comments will be placed throughout the program. Deficiencies in the standards might also indicate where defects are likely to occur in the code. For example, the standards might not place a limit on the size of program modules. In this light, auditors might expect that some modules will be large, inherently complex, and therefore error prone.

Understand the Program Specifications

By understanding the program specifications, auditors can address the question. Does the program do what it is supposed to do? Here they must make a choice about how they will obtain an understanding of the program specifications. One alternative is to review the documented program specifications those used by the programmer as the basis for constructing the code. By reviewing these specifications, auditors can check the correspondence of the code with the specifications. Using this approach, deficiencies in the specifications might also become apparent; for example, auditors might identify an important control that is missing in the specifications.

Obtain Source Code

When auditors obtain the source code that they wish to review, they must take steps to ensure that it is the current version and not outdated. Otherwise, auditors might not identify important defects in the source code for example. Un-authorized code that has been introduced to enable a defalcation to be perpetrated.

Review Programming Language Used

From the auditor's viewpoint, unfortunately a large number of programming languages now exist. They might often encounter a situation, therefore, in which they are not familiar with the programming language used to implement the source code they want to review. In these situations, auditors must either acquire familiarity with the programming language or rely on someone else to undertake the code review on their behalf. If auditors must rely on someone else to undertake the review and they conclude they cannot be confident in the integrity of the information that will provided to them, they must choose a technique other than code review to gather evidence.

Review Source Code

Currently, we have little theory or empirical evidence to help us choose the most effective and efficient way of reviewing program code. Many questions remain unanswered. Do some ways of reviewing code identify more errors than others? Are some ways of reviewing code faster than others? Does the best way depend on how the code is written (structured)? Should code be reviewed differently if, say, efficiency is the main concern rather than, say, data integrity? Does the effectiveness and efficiency of a code review technique depend upon our psychological and demographic characteristics?

Formulate Flaw Hypotheses

If auditors do not identify any defects in the source code that they review, most likely they can conclude that they can place reliance on the code to meet control objectives. In this light, auditors should be able to conclude

also that they can reduce the extent of subsequent substantive testing they undertake.

If auditors believe they have identified a defect in the code they have reviewed, however, two alternatives are then available. First, auditors might conclude that they do not wish to place reliance on the code in meeting control objectives. Accordingly, auditors should expand the extent of subsequent substantive tests to determine how the defect they believe exists has impacted the attainment of control objectives. Second, auditors might conclude that they need to investigate the presumed defect further. They should then formulate flaw hypotheses that predict the impact of the defect on control objectives. Auditors can then use some other evidence-collection technique to determine whether the predicted defect has the anticipated effect on control objectives. For example, they can design and use test data to test their flaw hypotheses.

Benefits and Costs of Code Review

The primary benefit of reviewing program source code is that it provides a level of detailed knowledge about a program that auditors will find difficult to acquire using other evidence-collection techniques. With other evidence-collection techniques, inferences must be made about the quality of the code on the basis of some test result. With program source-code review, however, auditors examine the code directly. In this light, they can obtain high levels of assurance about the quality of the code if they cannot identify any defects. Alternatively, if auditors do identify a defect, they can often make fairly precise predictions about' the impact of this defect on the financial statement assertions or effectiveness and efficiency assertions made by management.

2.4 Test data

The use of a sample of data to assess the quality of a program is fundamental to many evidence-collection techniques. It is based on the premise that it is possible to generalize about the overall reliability of a program if it is reliable for a set of specific tests.

As with program code review, it is unlikely that auditors will be interested in testing an entire program. Rather, they will focus on testing those parts of a program that they deem to be material and on which they intend to rely. Like program code review, test data can be an expensive evidence-collection procedure. Auditors need to focus their efforts, therefore, on those parts of a program where the payoffs will be highest.

In the following subsections, we first examine the nature of reliable test data. We then discuss some major approaches to the design of test data that auditors might employ. Finally, we examine some ways that auditors can create test data and how they might use various automated tools to assist them in the design, creation, and use of test data.

2.4.1 Nature of Reliable Test Data

As background to our discussion on test data techniques, let us examine first the nature of reliable test data. Perhaps the most widely accepted notion of reliable test data has been proposed by Gerhart and Goodenough and Howden. They argue as follows. Suppose we have a program, *P,* for computing a function, *F,* whose domain (input) is a set, *D.* Let $T \subset D$ be the test data used to determine whether *P* contains any defects. Tis deemed to be a reliable set of test data for *P* if:

$$P(x) = F(x) \forall x \in T \rightarrow P(x) = F(x) \forall x \in D$$

In other words, *T* is reliable if it reveals an error in *P* whenever *P* contains an error. The fact that *P* has no defects when it processes every element in the set of test data is sufficient to ensure that *P* will have no defects when it processes *any* element in its input domain.

2.4.2 Approaches to Designing Test Data

Because we cannot automatically generate a reliable set of test data, test data therefore must be designed. To improve the quality of the test-data design, we should use a systematic approach. Otherwise, important defects in the program might be missed. Moreover, excessive test data might be designed and used. We must avoid the tendency to believe that more comprehensive testing will result when a larger amount of data is designed, created, and executed through a program. Unfortunately, a larger amount of test data ensures neither that a program's critical features will be tested, nor that testing will be carried out in the most economical manner. Auditors are often subject to severe cost constraints. In this light, their use of systematic approaches to the design of test data, therefore, is essential.

2.4.3 Black-Box Test-Data Design Methods

Auditors use black-box test-data design methods whenever their primary focus is on whether some part of an application system meets its functional requirements. Besides functional defects, however, black-box testing might also help to identify other types of defects in programs: *(1)* user interface errors, *(2)* errors in interfacing with external systems or databases, *(3)* efficiency problems, *(4)* initialization errors, and *(5)* termination errors.

2.4.4 White-Box Test-Data Design Methods

Auditors use white-box test-data design methods whenever their primary focus is on whether defective execution paths exist in a program. The underlying assumption is that significant information about any defects in a program can be obtained by systematic execution of different logic paths through the program. Furthermore, when auditors use white-box test-data design methods, they are recognizing that predicting the actual behavior of

a program is still difficult even when they have substantial prior information about the program.

As with black-box testing, auditors begin white-box testing by choosing those parts of a program that they deem to be material from the viewpoint of their audit. Next auditors study a source-code listing of the program to identify where these parts have been implemented within the program code. Their primary goal is to identify the *control structure* underlying the code because the control structure indicates the different execution paths through the program. Having identified the critical sections of source code and the control structure, auditors can then proceed to design test data to traverse this code.

2.4.5 Creating Test Data

When auditors have designed the test data they need for evidence-collection purposes, their next step is to create test data that complies with the design. This task can be difficult and time-consuming to complete, especially when auditors must prepare a large number of test cases.

One approach auditors can use to reduce the amount of resources required to create test data is to rely on existing production data. They should not simply select a random sample of this production data and execute it through the code on which they intend to rely. This approach is unlikely to lead to effective and efficient testing. Rather, auditors should select instances of production data that comply with their test-data design. In this light, they might use generalized audit software, for example, to select cases off the production files that satisfy their design. If there are no production cases that satisfy the design, auditors might still select cases that almost comply with the design and modify them appropriately.

Whenever auditors use production data as the basis for test data creation, however, they must take care that they do not overlook logic that is exercised infrequently by production data and therefore might be missing from the production files they use as the basis for creating their test data. By ensuring that they check the correspondence between test-data design and the production data they have selected, auditors reduce the chances of failing to exercise important logic when they use production data.

Auditors might also be able to use test data prepared by the auditee to assist in the creation of their own test data. The auditee's information systems personnel should have prepared test cases to test the program that is the focus of the evaluation. Again, auditors can select test cases off the auditee's test data files that comply with their design using, say, generalized audit software. Once more, auditors must be diligent in checking the correspondence between the set of test cases they choose and their test-data design.

Sometimes, also, the auditee will prepare test data for an application system in cooperation with the auditor. The objective is to prepare a comprehensive "base case" that can be used to test the application'system

before it is released into production and during subsequent releases after maintenance work has been carried out to the application system. Whenever auditors participate in the development of test data for *base case system evaluation* (BCSE), production test data is more likely to be useful as a means of accomplishing testing objectives.

New test data must be created if cases that satisfy an auditor's test-data design are not available from either existing production files or the auditee's test data files. Various types of automated aids are available to assist with test. data creation (see the next section). Parameter values must be provided as input to these aids—for example, the range of values to be generated for a particular data item. These parameter values should not be chosen haphazardly. Rather, they should be chosen carefully on the basis of the test-data design.

2.4.6 Automated Aids to Support Design, Creation, and Use of Test Data

Because design, creation, and use of test data are often time-consuming, resource intensive, and error prone, various automated aids have been developed to assist in the completion of these tasks. Auditors might find the following aids to be valuable when undertaking evidence-collection work using test data:

To illustrate how these tools might be used, assume auditors have employed decision tables to facilitate their design of test data using the equivalence partitioning approach. Recall that equivalence partitioning is a black-box testing approach; that is, it is based on the program's specifications rather than direct examination of the program's source code to identify its control structure. The risk with equivalence partitioning, therefore, is that auditors will not execute sections of code because they are not documented in the specifications.

Automated Aid	*Overview*
Test data/file generators	These tools allow test cases and test files to be generated automatically. For example, auditors can specify the boundary value for a conditional statement, and a test data generator will generate three test cases: one with a value equal to the boundary value, one with a value just exceeding the boundary value, and one with a value just less than the boundary value.
Test capture/playback tools	These tools capture live transactions and allow auditors to reuse them for testing. They were developed to assist programmers, because it is often difficult to re-create the conditions that lead to failure in an online system. Auditors can use them to capture streams of data that often lead to program failure—for example, high-volume, stress conditions. The input test streams can also be edited to examine the performance of the program under changed conditions.
Test coverage/execution path monitor tools	These tools take source code as input and insert flags in the source code to indicate which statements and branches have been traversed by test data. The instrumented program is then compiled, and test data is then executed using the compiled program. At the completion of a test run, the instrumented program will print a report showing which statements or branches have or have not been executed.
Test drivers/harnesses	These tools facilitate the overall testing process. They act as an intermediary between the operating system and the program to be tested. If the program being tested terminates abnormally, they will capture and print useful diagnostic information. At the same time they will prevent disruption to or termination of the operating system. They also facilitate top-down testing by allowing modules that have not yet been written to be called. They will print a message showing the module has been called and perhaps pass parameter values back to the calling module. When a program reaches a certain point in its execution, they will either halt the program conditionally or unconditionally and permit programmers to access useful data (e.g., the contents of a record).
Test output comparators	These tools allow the results obtained from executing one set of test data to be compared with the results obtained from executing another set of test data. Differences between the two sets of results are highlighted. Comparisons can also be made between the results obtained from executing the same production data through two versions of a program.
Static analyzers	These tools assist in the design of test data. They will read program source code and identify alternative ways of reaching different sections of code. Test data can then be designed to traverse each of these alternative paths. They will also identify sections of code that cannot be reached.

After auditors have designed and created the test data, they might then instrument the program using an execution path monitor tool. They can then execute the instrumented program using the test data they have created. If the execution path monitor identifies sections of code they have not tested, three possibilities exist: *(1)* their test data is erroneous or incomplete; *(2)* the program code is erroneous; *(3)* the program code does not comply with the specifications. Whatever the reason, auditors can then go back and either modify their test data or investigate the program further to detect the source of the discrepancy.

2.4.7 Benefits and Costs of Test Data

The major benefit of using test data as an evidence-collection technique is that it allows auditors to examine the quality of program code directly. Well-designed test data specifically addresses the question of whether the code complies with specifications. The quality of the code need not be inferred from the quality of production data that the program has processed. Well-designed test data also specifically addresses the question of whether defective execution paths exist in the code.

Some people claim that a major benefit of using test data arises from auditors needing little technical competence with computers to use the technique. For a simple batch system or an undisciplined approach to the use of test data, this claim could be true. As we have seen in this chapter, however, the design and creation of high-quality test data and the use of automated tools to support the test data approach require auditors to have substantial knowledge of information technology.

The primary disadvantage of using test data as an .evidence-collection technique is that it is often time-consuming and costly. As the theory underlying test-data design improves and automated tools to support test-data design, implementation, and use become more widely available, this disadvantage might become less important. As the quality of code improves and more code is implemented via high-level languages, use of test data also becomes more feasible.

2.5 Program code comparison

Auditors use program code comparison for two reasons. First, it provides some assurance that they are auditing the correct version of software. For example, they might wish to undertake code review of the material parts of a program. They need to determine, therefore, whether the source code provided to them for review purposes corresponds to the source code used to compile the production object code. Using program code comparison, auditors can determine whether the two versions of the source code are the same. We discuss this use of code comparison next.

Second, it provides some assurance that any software used as an audit tool is the correct version of the software. For example, assume an organization has used specialized audit software to collect audit evidence

throughout some period of time. Auditors need to determine whether they can rely on the evidence as a basis for forming their audit judgments. In this light, auditors might wish to compare an audit version of the specialized audit software (one that they have tested comprehensively at a prior time and one that they have judged to be free of material defects) with the organization's version of the software. If auditors find correspondence between the audit version and the organization's version of the software, they might then conclude they can rely on the evidence collected by the specialized audit software as the basis for the audit judgments they make.

2.5.1 Types of Code Comparison

Software is available to undertake two types of program code comparison: *(1)* source-code comparison and *(2)* object-code comparison. With source-code comparison, the software should provide a meaningful listing of any discrepancies between two versions of a program's source code. Nevertheless, often auditors must obtain further assurance that the source-code version they are evaluating is the one used to compile the production object code.

With object-code comparison, auditors will have difficulty identifying the *nature* of any discrepancies found between two versions of object code. Recall that few people can read and understand object code. Thus, a report of discrepancies is unlikely to provide much assistance. Instead, object-code comparison is better used to obtain an answer to the following simple question: Are there any discrepancies between two versions of object code? If discrepancies are identified, other techniques must be used to identify the nature of and cause of the discrepancies.

2.5.2 Using Code Comparison

Source-code and object-code comparison are often most effective as audit techniques when auditors use them in conjunction with each other. Figure 2-2 shows an overall approach we can follow. First, we compare the audit version of a program's source code with that version that the organization we are auditing contends is the source code used to compile the production object code. Any discrepancies identified between the source-code versions must be reconciled. Second, we compile either the audit or organizational version of the source code with the compiler used to produce the production object code. Third, we compare the object code produced via this compilation with the production version of the object code. Any discrepancies identified mean either the wrong compiler has been used or the organization has supplied us with the wrong version of the source code or production object code.

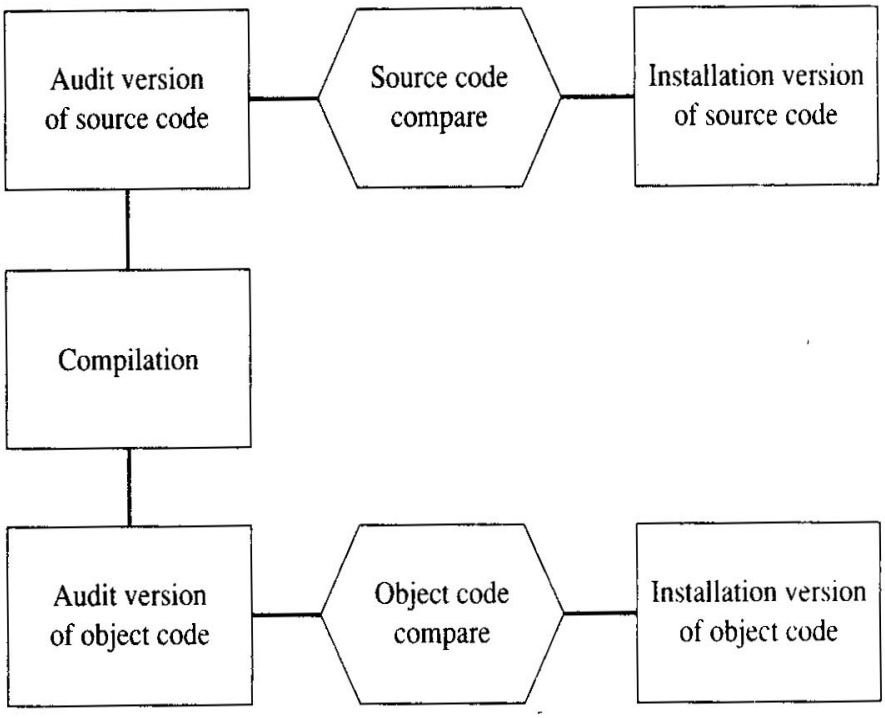


FIGURE 2-2 Use of code comparison for evidence gathering

2.5.3 Benefits and Costs of Code Comparison

The code comparison technique is an easy way of identifying changes made to programs. The software that performs code comparison usually is neither costly to purchase nor to execute. Furthermore, auditors require little technical skill to be able to use the software.

Identifying the implications of any discrepancies found in the code, however, requires some knowledge of programming and the programming language used. Auditors must be able to determine whether a discrepancy reflects a material change to the program or a minor change. They must also he able to read and understand the program code if they are to be capable of making an informed judgment on the materiality of the discrepancy.

A limitation of the technique is that it does not provide any evidence directly on the quality of the code being compared. Auditors do not know, for example, whether the code safeguards assets or is efficient. They must first thoroughly evaluate one version of the code against audit objectives and then use this version as a blueprint for comparison purposes. If auditors identify discrepancies between another version of the code and this blueprint version, they must then be able to judge whether the changes made have undermined the quality of the code.

Code comparison programs also differ in terms of the quality of their output. For example, assume two versions of code differ because a block of documentation text in the program has been shifted from one position in the program to another position in the program. Some code comparison programs will show that the entire block has been changed. They will print out the entire block as having been deleted from one part of the code. They will also print out the entire block as having been inserted in another section of the code. Other code comparison programs will show that the block of text simply has been shifted in terms of its position in the program. If auditors were to frequently use the former type of code comparison program, eventually, they might become careless and miss a material change that was made in a block of code that had also been shifted within the program they arc evaluating.

3. Concurrent Auditing Techniques

Structure

Objectives

After going through this unit, you should be able to:

- understand to Basic nature of concurrent auditing techniques
- understand to Need for concurrent auditing techniques
- understand to Types of concurrent auditing techniques
- understand to Implementing concurrent auditing techniques
- understand to strengths/limitations of concurrent auditing techniques

3.1 Introduction

In this lesson, we examine the basic nature of concurrent auditing techniques, the reasons why they were developed, their relative advantages and disadvantages, and some methods of implementing concurrent auditing techniques. A large number of different concurrent auditing techniques have now been developed. A close examination, however, reveals that they are all variations on a theme. For this reason, here we cover just a few of the major techniques that have been used. If we understand the nature of these few techniques, we should then be able to adapt them in various ways to suit the particular needs of any audit we might wish to undertake.

3.2 Basic nature of concurrent auditing techniques

Concurrent auditing techniques use two bases for collecting audit evidence. First, special audit modules are embedded in application systems or system software to collect, process, and print audit evidence. Second, in some cases, special audit records are used to store the audit evidence collected so auditors can examine this evidence at a later stage. These records can be stored on application system files or on a separate audit file.

Though evidence collection is concurrent with application system processing, the timing of evidence reporting is a decision that auditors can make. If a concurrent auditing technique identifies a critical error or irregularity, auditors might program the embedded audit routines to report the error or irregularity immediately. In this light, the evidence could be transmitted directly to a printer or terminal in the auditor's office (Figure 3-1a). In other cases, however, immediate reporting of the error or irregularity might not be essential. Auditors can then store the evidence for reporting at some later time (Figure 1-1b).

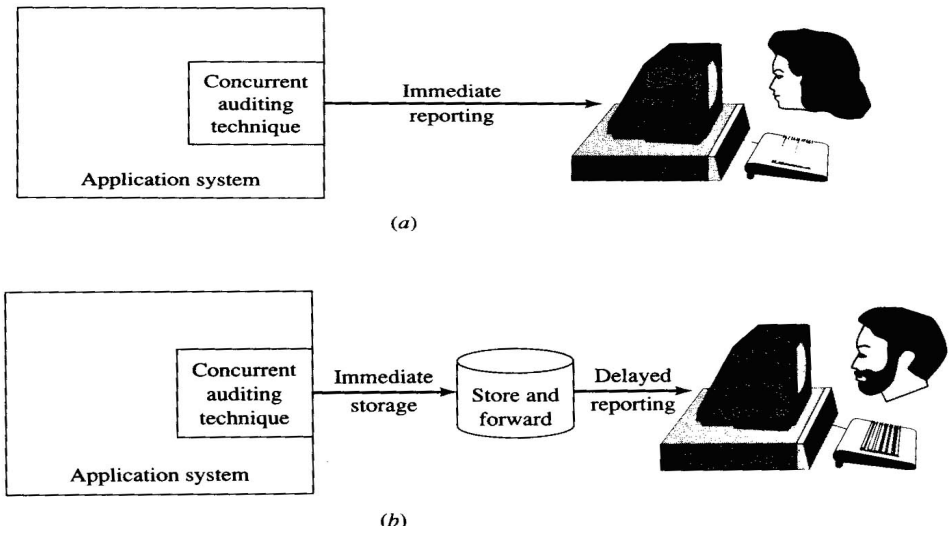


FIGURE 3-1 (a) Immediate reporting of event with concurrent auditing, (b) Delayed reporting of event with concurrent auditing.

3.3 Need for concurrent auditing techniques

Concurrent auditing techniques were developed in the late 1960s and early 1970s to address a set of problems that were arising as computer-based information systems became more widespread. Moreover, several recent trends provide further impetus to deploy concurrent auditing techniques more extensively. We examine these factors briefly in the following subsections.

3.3.1 Disappearing Paper-Based Audit Trail

Historically, auditors have placed substantial reliance in evidence-collection work on the paper trail that documents the sequence of events that have occurred within an information system. This trail has provided a means of determining whether material errors or irregularities have occurred in an information system.

Paper-based audit trails have been progressively disappearing, however, as computer systems have replaced manual systems and as source documents have given away to screen-based input and output. A paper audit trail is no longer left automatically as the outcome of an event in an application system. Instead, the existence of an adequate audit trail depends on purposeful design and implementation of processes to record events in computer systems.

3.3.2 Continuous Monitoring Required by Advanced Systems

A common characteristic of advanced information systems is that they are tightly coupled; that is, an event in one application system leads to an event in another application system. For example, consider the set of systems shown in Figure 3-2. The receipt of an order from a customer leads automatically to the issuance of a production order. This order is then exploded via a bill-of-materials application system to determine the parts and labor needed to fulfill the order. An inventory system next checks the availability of inventory, reserves the inventory if it is available, and places an order on a supplier via an electronic data interchange system if any part is in short supply. A production scheduling system then schedules the order for production. When production is to be started, a production control system initiates the issue of materials and labor requisitions and collects production performance and costing data as the order progresses through the production system. When production of the goods is completed, a finished goods system updates inventory. Finally, a shipments system initiates dispatch of the goods to the customer.

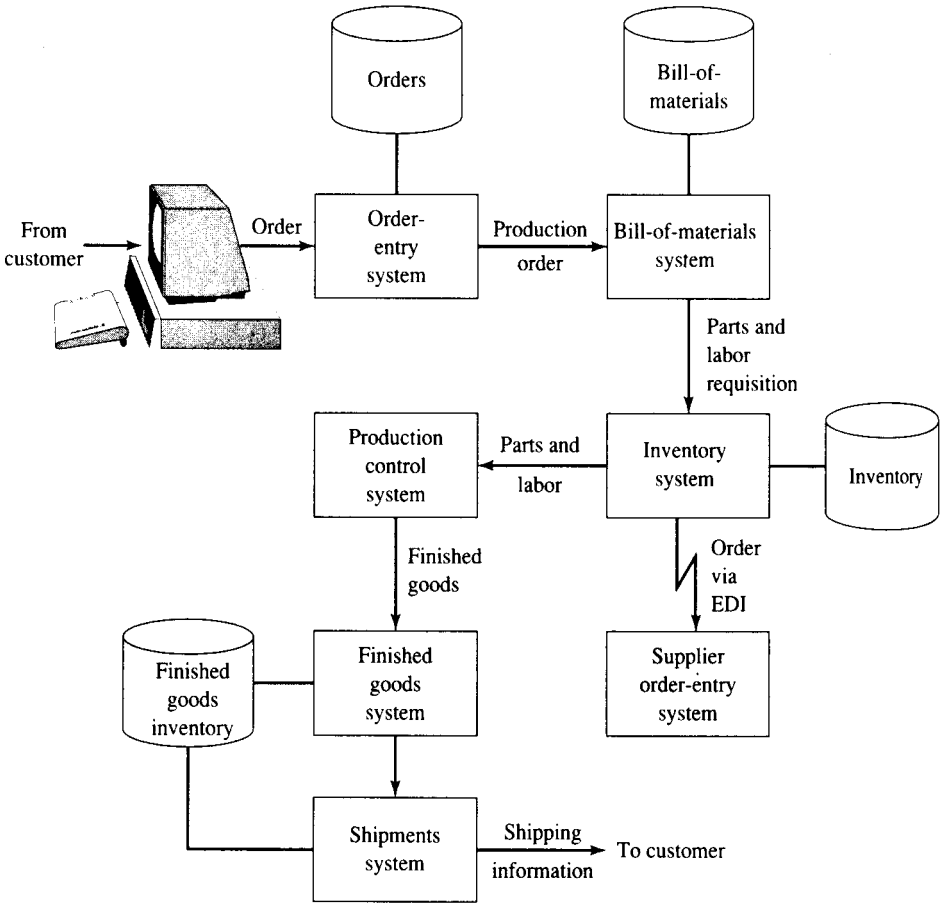


FIGURE 3-2 Example of tightly coupled application systems.

When systems are tightly coupled, errors or irregularities in one system can quickly propagate to other systems and cause material losses. For example, assume an order-entry program corrupted data in an order in the set of systems described previously. If the corrupted data resulted in an overstatement of the quantity required by the customer, the bill-of-materials system would in turn overstate the quantities of materials and labor required. Excessive raw materials might then be ordered as inventory falls below reorder levels, and additional labor might be hired unnecessarily. Subsequent orders might be lost because the inventory system indicates insufficient materials and labor are available to fill these orders on a timely basis. Limited production capacity might be wasted as goods are produced that no one has ordered. Further costs might be incurred as excessive goods are shipped to the customer and presumably eventually returned by the customer.

To mitigate losses when systems are tightly coupled, timely identification of errors or irregularities is essential. For this reason, auditors might use concurrent auditing techniques to report any event that could manifest an error or irregularity immediately it occurs. They can then follow up on the event to determine whether it represents a material exposure.

3.3.3 Increasing Difficulty of Performing Transaction Walkthroughs

Auditors often gain an understanding of an application system by taking typical transactions and tracing them through the various execution paths that can be traversed within the system. Walking typical transactions through a system also helps to identify the system's strengths and weaknesses and plan subsequent audit tests.

Advanced information systems make the walkthrough process more difficult because they often have a large number of complex execution paths. As demonstrated previously, extensive coupling between different application systems also complicates matters. For example, understanding how a parts master file is updated might mean auditors must examine processes in the production scheduling system, inventory reordering system, purchasing system, receiving system, and warehousing system.

Concurrent auditing techniques facilitate our understanding of advanced systems by collecting all the information normally obtained from a walk-through in the one place. They can be used to capture images of a transaction as it traverses a particular execution path. These images then can be written to a file for subsequent examination. When auditors then attempt to understand a system and to identify its strengths and weaknesses, all the information associated with the different execution paths in the system exists in the one place.

3.3.4 Presence of Entropy in Systems

All systems have characteristic called *entropy,* which is the tendency of systems toward internal disorder and eventual collapse. In information systems, entropy arises in a variety of ways. One form occurs because user information requirements change as the business they undertake changes. As a result, existing information systems become less effective at meeting their needs. A second form occurs through increases in the numbers of transactions that must be processed. The existing hardware and software eventually becomes unable to handle the workload satisfactorily. Errors then occur, for example, because transaction queues exceed their maximum allowed length. A third form arises through having to maintain existing systems. Knowledge about existing systems gradually degrades over time as the personnel who developed these systems leave to take other positions. As a result, errors creep into systems during maintenance work because maintenance staff does not fully understand the systems on which they are working.

Concurrent auditing techniques provide a means of identifying increasing entropy in information systems at an early stage. They can be used to gather data on error and exception frequencies and to give advance warning of stresses being placed on systems. Thus, they facilitate auditors' understanding of a system's evolution. They also assist the stakeholders in an information system to mitigate the consequences of errors by providing feedback that allows them to undertake timely modifications in light of changing circumstances.

3.4 Types of concurrent auditing techniques

Although many concurrent auditing techniques have been developed, Mohrweiss argues that they all fall into three categories: *(1)* those that can be used to evaluate application systems with test data while they undertake production processing, *(2)* those that can be used to select transactions for audit review while application systems undertake production processing, and *(3)* those that can be used to trace or map the changing states of application systems as they undertake production processing. With this classification in mind, we examine four major concurrent auditing techniques in the following subsections:

1. Integrated test facility (ITF),
2. Snapshot/extended record,
3. System control audit review file (SCARF), and
4. Continuous and intermittent simulation (CIS).

The ITF technique can be used to test an application system with test data during normal production processing. The snapshot/extended record technique can be used to trace the changing states of an application system as it undertakes production processing. The SCARF and CIS techniques can be used to select transactions during production processing for audit review.

3.4.1 Integrated Test Facility

The ITF technique involves establishing a minicompany or dummy entity on an application system's files and processing audit test data against the entity as a means of verifying processing authenticity, accuracy, and completeness. For example, auditors could use ITF in the following ways: If the application is a payroll system, they might set up a fictitious person in the database; if the application is an inventory system, they might set up a fictitious stock item in the database; if the application is an EDI system, they might work cooperatively with auditors in other organizations and set up dummy entities in the database of their own organization or client organization as well as the databases of other organizations with which their own organization or client organization interacts. Auditors would then use test data to update the fictitious entities. This test data would be included with the normal production data used as input to the application system.

Using ITF involves two major design decisions:

1. What method will be used to enter test data?
2. What method will be used to remove the effects of ITF transactions?

3.4.2 Snapshot/Extended Record

For application systems that are large or complex, tracing the different execution paths through the system can be difficult. If auditors wish to perform transaction walkthroughs, therefore, they could face a difficult or impossible task. A simple solution to the problem is to use the computer to assist with performing transaction walkthroughs.

The snapshot technique involves having software take "pictures" of a transaction as it flows through an application system. Typically auditors embed the software in the application system at those points where they deem material processing occurs. The embedded software then captures images of a transaction as it progresses through these various processing points. To validate processing at the different snapshot points, auditors usually have the embedded software capture both beforeimages and afterimages of the transaction. They then can assess the authenticity, accuracy, and completeness of the processing carried out on the transaction by scrutinizing the beforeimage, the afterimage, and the transformation that has occurred on the transaction. Figure 18^t shows how the technique might be used to obtain audit evidence at various points in a simple batch system.

Implementing the snapshot technique requires auditors to make three major decisions. First, they must decide where to locate the snapshot points within the application system that is the focus of the audit. Auditors should make this decision on the basis of the materiality of the processing that occurs at each point in the application system. In some circumstances, however, auditors might have to temper their desire to capture snapshots in light of demands placed on the application system. For example, a processing point might be material, but efficiency considerations might be paramount. If auditors embed software to capture snapshots at this point, they might produce an unacceptable degradation in response times when the system is under load.

The second decision auditors must make is when they will capture snapshots of transactions. Auditors might have the embedded software always make snapshots for certain high-exposure transactions—for example, transactions in a financial institution that alter the terms of major loans. Alternatively, they might choose particular transactions for scrutiny via snapshot before they are entered into the application system. They must then tag these transactions in some way, and the embedded software must recognize that the transactions are tagged for snapshot purposes. Auditors might also program the embedded software to make snapshots of various transactions based on some type of sampling plan. Whatever the approach auditors use, they must be careful to obtain sufficient, reliable evidence but not to capture so much evidence that they suffer from information overload.

The third decision auditors must make relates to reporting of the snapshot data that is captured. The embedded software must provide sufficient identification and timestamp information for each transaction to enable auditors to determine the transaction to which the snapshot data applies, the sequence of state changes that has occurred as the transaction has passed through the various snapshot points, the processing points for which the snapshot data has been captured, and the time and date at which the snapshot data for each processing point was captured. A reporting system must also he designed and implemented to present this data in a meaningful way.

A modification of the snapshot technique is the *extended record technique.* Instead of having the software write one record for each snapshot point, auditors can have it construct a single record that is built up from the images captured at each snapshot point. This record is progressively built

as the transaction that is of interest to auditors traverses the various snapshot points in the application system. Extended records have the advantage of collecting all the snapshot data related to a transaction in one place, thereby facilitating audit evaluation work.

3.4.3 System Control Audit Review File

The system control audit review file (SCARF) technique is the most complex of the four concurrent auditing techniques we will examine. It involves embed-ding audit software modules within a host application system to provide continuous monitoring of the system's transactions. These audit modules are placed at predetermined points to gather information about transactions or events J within the system that auditors deem to be material. The information collected j is written onto a special audit file the SCARF master file. Auditors then examine the information contained on this file to see if some aspect of the application system needs follow-up. Figure 18-6 illustrates the method as applied to a master file update program.

In many ways, the SCARF technique is like the snapshot/extended record technique. Indeed, the SCARF embedded software can be used to capture snapshots and to create extended records. We see subsequently, however, that other types of data can be collected via the SCARF embedded modules for example, system exceptions deemed material. Moreover, we also see subsequently that the SCARF technique uses a more complex reporting system than the snapshot and extended record techniques.

Using SCARF involves two major design decisions:

1. Determining what information will be collected by SCARF embedded audit routines, and
2. Determining the reporting system to be used with SCARF.

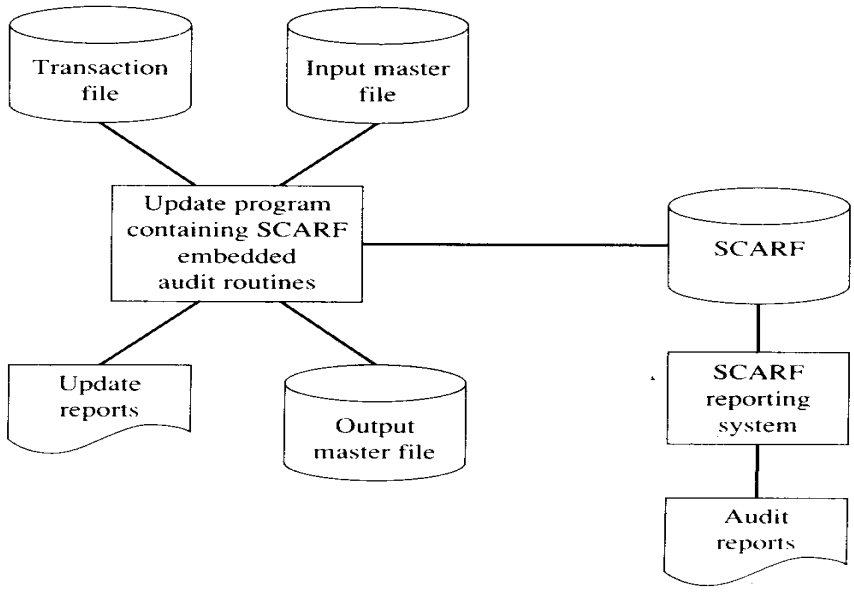


FIGURE 3-3 Use of SCARF with a master file update program.

3.5 Implementing concurrent auditing techniques

When auditors implement concurrent auditing techniques, they should follow the same steps necessary to achieve any well-implemented system. Because these steps have been described extensively in Chapters 4 and 5, the following sections provide only a brief overview and highlight those aspects having special relevance for concurrent auditing.

3.5.1 Perform a Feasibility Study

Concurrent auditing techniques result in overheads for application systems he-cause special audit records and audit routines must be embedded within them. Sometimes these overheads can be minor. For example, response times and turnaround times increase only marginally. Sometimes they can be unacceptable. For example, system response times degrade to the point where users are seriously impaired in carrying out their work.

Auditors must always consider carefully the costs and benefits of using concurrent auditing techniques. The costs include development costs, implementation costs, maintenance costs, and operations costs. When determining operations costs, auditors must take into account the externalities caused by concurrent auditing techniques, such as the overhead costs imposed on the host application system and the resulting impact on users. When determining benefits, auditors should recognize that the work carried out by several groups of stakeholders might be facilitated by the existence of concurrent auditing techniques. Beside the benefits obtained for the audit, information systems staff, for example, might benefit because concurrent auditing techniques facilitate testing of application systems.

If auditors can implement concurrent auditing techniques when an application system is first developed, the costs are likely to be lower. The system can then be designed from the outset to accommodate the concurrent auditing techniques. If auditors have to modify an existing application system to accommodate concurrent auditing techniques, however, often the costs will be high. In this light, auditors should seek to participate in the development of new application systems to determine whether concurrent auditing techniques should be incorporated in them.

3.5.2 Seek the Support of Groups Affected by Concurrent Auditing

Because concurrent auditing techniques require ongoing support, they are typically the responsibility of an organization's internal audit staff. Nevertheless, the organization's external auditors should also be contacted as they might have requirements that can be met by concurrent auditing techniques. In any event, they must be apprised of how concurrent auditing techniques are used within the organization. Use of concurrent auditing techniques might mean that external auditors can place greater reliance on the work carried out by internal auditors. Moreover, if the techniques are used properly, external auditors are likely to assign a lower control risk to the organization and decrease the extent of substantive testing accordingly.

3.5.3 Ensure that the Relevant Expertise Is Available

If auditors are to use concurrent auditing techniques successfully, they must have a reasonable level of expertise in information systems audit and control. For a start, auditors must be able to identify where material processing points occur within application systems. Otherwise, they will not be able to determine where embedded audit modules are best placed. Auditors must also be able to make astute decisions in relation to such matters as the record structures and reporting systems that concurrent auditing techniques will use. Otherwise, they might not collect the evidence they require, or they might not be able to obtain the evidence in a form that is useful to them. Finally, auditors need the expertise either to implement, operate, and maintain concurrent auditing techniques or to evaluate the work of others who undertake these tasks on their behalf.

3.5.4 Ensure the Commitment of Stakeholders

Concurrent auditing techniques require resources to develop, implement, operate, and maintain. Auditors require management's commitment; therefore, if the resources needed to support concurrent auditing techniques are to be available on an ongoing basis. Management should be committed if they can see clear payoffs from using concurrent auditing techniques. The onus is on auditors, therefore, to use concurrent auditing techniques effectively and to demonstrate clearly that benefits exceed costs.

3.5.5 Make the Necessary Technical Decisions

When auditors implement concurrent auditing techniques, recall that they must make several key technical design decisions. For ITF, they must choose the test data method to be used and the method of removing the effects of the ITF transactions. For snapshot/extended records and SCARF, they must decide on those points in the system where data will be captured and the type of data that will be captured. In the case of SCARF, they must also determine the structure of the reporting system. For CIS, they must choose the transactions whose application system processing will be simulated.

3.5.6 Plan the Design and Implementation

When the necessary technical decisions have been made, auditors can proceed with the specific design for the concurrent auditing techniques. They can also plan the implementation of the concurrent auditing techniques. Decisions must be made, for example, on the data structures to be used, the programming language to be used, the ways in which audit modules will be embedded within the host application systems, and the methods to be used to protect the integrity of the audit modules.

3.5.7 Implement and Test

The implementation of concurrent auditing techniques will be facilitated if high-level software is available to support use of concurrent auditing. For some types of hardware/software platforms, for example, software has been developed to initiate calls to a library of concurrent auditing routines,

receive information back from these routines, and process and present the information in a form that facilitates audit work.

3.5.8 Postaudit the Results

After concurrent auditing techniques have been running for some time, auditors should evaluate their costs and benefits, particularly in light of the estimates of costs and benefits they made during the feasibility phase. This postaudit identifies defects that possibly can be corrected or ways in which the techniques can be used more effectively and efficiently. It also-can lead to the conclusion that concurrent auditing techniques should be scrapped.

A postaudit also formalizes the experience gained with concurrent auditing techniques and establishes guidelines for their design and implementation in other application systems. Auditors are then better placed to make decisions on whether the techniques should be implemented elsewhere and, if so, how they should be implemented.

3.6 Strengths/limitations of concurrent auditing techniques

In spite of the apparent advantages of concurrent auditing techniques, their use has not been widespread. Periodically, surveys have been undertaken of the extent of their use. The results of these surveys are problematical for several reasons. First, they have not used common definitions for the different types of concurrent auditing techniques. Second, in some cases, data on a particular technique has not been collected. Third, different types of auditors have been surveyed. Nonetheless, a common theme emerges: Use of concurrent auditing techniques has been fairly stable and limited over many years. The following table summarizes the results of several of these surveys, where the percentages in the table reflect the number of respondents who indicated they had used the concurrent auditing technique:

Technique \ *Study*	*IIA (1977)*	*Perry and Adams (1978)*	*Tobinson and Davis (1981)*	*Langfield-Smith (1987)*	*Mohrweis (1988)*	*IIA (1991)*
Integrated Test Facility	5.0%	15.0%	13.3%	22.7%	12.2%	11.0%
Embedded Audit Modules/SCARF	15.8%	20.0%	13.3%	2.1%	11.9%	11.0%
Snapshot/Extended Records	18.4%	20.0%	4.4%	—	9.9%	—

The results of these surveys also indicate that several factors affect the use of concurrent auditing techniques:

1. Internal auditors are more likely to use concurrent auditing techniques than external auditors. This situation occurs because internal auditors

should be able to obtain the resources required from their organizations to support the development, implementation, operation, and maintenance of concurrent auditing techniques.

2. Concurrent auditing techniques are more likely to be used if auditors are involved in the development work associated with a new application system. As discussed previously, it is easier to install concurrent auditing techniques at the outset rather than to retrofit an application system with these techniques.

3. Concurrent auditing techniques are more likely to be used if auditors employ other computer-based audit techniques. In short, auditors need the knowledge j and experience of working with computer systems to be able to use concurrent auditing techniques effectively and efficiently.

4. Concurrent auditing techniques are more likely to be used as the incidence of i automatically generated transactions in application systems goes up. The audit trail is less visible for these types of transactions, and the costs of errors and irregularities associated with them can be high.

Limitations of concurrent auditing techniques:

1. The costs of developing, implementing, operating, and maintaining concurrent auditing techniques can be high. For this reason, the benefits of using concurrent auditing techniques must be clear to all stakeholders. Otherwise, the needed resources and support will not be forthcoming.

2. Unless we have substantial knowledge of and experience with information systems auditing, it is unlikely that we will be able to use concurrent auditing techniques effectively and efficiently. Moreover, we must have a good understanding of the target application system if we are to be capable of placing concurrent auditing techniques at strategic points within it.

3. Concurrent auditing techniques are unlikely to be effective unless they are implemented in application systems that are relatively stable. If the host application system is changing frequently, the costs of maintaining concurrent auditing techniques are likely to be high. We might be able to justify these costs on the basis of the high exposures associated with the high levels of volatility in the application system. Nonetheless, we must be aware of the increased difficulties we are likely to face.

In summary, assessing the benefits and costs of using concurrent auditing techniques can be difficult. For a start, identifying all the benefits and costs associated with their use is hard. Because several different sets of stakeholders exist, we must take care in assessing how concurrent auditing techniques affect each of them. Valuing the benefits and costs is also difficult. Again, we need to consider value from the viewpoints of all the different stakeholder groups.

4. SUMMARY

There is a variety of software available to auditors to assist in evidence collection. For a start, auditors can use generalized audit software, which has been designed specifically to allow them to gain access to and manipulate data maintained on computer storage media. It provides powerful functions that enable access to files maintained in a variety of formats, sorting and merging of files, selection of data that satisfy certain conditions, statistical sampling and evaluation of data, arithmetic operations on data, stratification and frequency analysis of data, file creation and updating, and flexible reporting of results obtained.

Three evidence-collection techniques used primarily to evaluate the quality of program logic are code review, test data, and code comparison. Each can be used independently. Each can also be used in conjunction with the other two, however, to perform an integrated test of whether defects exist in a program. Code review provides a basis for auditors to generate flaw hypotheses about program logic. Test data enables auditors to test these hypotheses. Code comparison allows auditors to test whether the production version of a program used is the tested version.

Concurrent auditing techniques collect audit evidence at the same time as application system processing occurs. This evidence can be written to a file and periodically printed for auditors to analyze and evaluate. Alternatively, auditors can print or display the evidence immediately so they can determine whether to take some type of immediate action for example, commence investigation of a potential irregularity.

5. Questions

1. Briefly discuss the motivations for developing generalized software specifically for audit purposes. Even though generalized retrieval software already existed before audit software was developed, why did auditors prefer to develop their own software packages?

2. What is a generalized audit software package?

3. What purposes might auditors seek to achieve in using generalized audit software to examine the quality of data maintained on an application system files?

4. How can auditors use generalized audit software to examine the existence of entities that the data purports to represent?

5. Briefly explain how code review can he used to identity ineffective code and nonstandard code in a program.

6. On what criteria should auditors select the source code to be examined during program source code review?

7. Briefly explain the difference between the black-box approach to test-data design and the white-box approach to test-data design.

8. Briefly explain the nature of concurrent auditing techniques.

9. What is entropy? Information systems staff often has a very high rate of turnover in their jobs. Is this a form of entropy in information systems? If so, how can concurrent auditing techniques are used to help mitigate the effects of this form of entropy?

10. Give *two* advantages of using concurrent auditing techniques in an organization.

6. REFERENCE BOOKS

1. Weber R; Information Systems Control and Audit (Person Education)

2. Dube: Information systems for Auditing (TMH)

3. Auditing Information Systems, 2nd Edition. Jack J. Champlain (Wiley)

UNIT – V

1. Evaluating Asset Safeguarding and Data Integrity

Structure

Objectives

After going through this lesson, you should be able to:

- discuss measures of asset safeguarding and data integrity
- discuss about a nature of the global evaluation decision
- understand how to determinants of judgment performance
- discuss about Audit technology to assist the evaluation decision
- discuss about Cost-effectiveness considerations

1.1 Introduction

In this lesson, we examine both the decision on how well assets are safeguarded and the decision on how well data integrity is maintained. We consider these decisions jointly because substantial overlap exists in terms of the evaluation methodologies that auditors can use for each decision.

The lesson proceeds as follows. In the first section, we discuss various measures of asset safeguarding and data integrity that auditors can use as the basis for their judgment process. Next, we examine the nature of the global evaluation decision they must make. We then discuss those factors that audit research has shown are major determinants of the quality of auditors' judgments when they make the global evaluation decision. In the subsequent section, we examine various tools that have been developed to assist auditors make the global evaluation decision. Finally, we discuss how auditors might evaluate the cost-effectiveness of controls.

1.2 Measures of asset safeguarding and data integrity

To evaluate how well assets are safeguarded and data integrity is maintained, auditors need some kind of measurement scale. Asset safeguarding and maintenance of data integrity are not all-or-nothing affairs; assets are safeguarded and systems maintain data integrity to varying degrees. Auditors need to be able to evaluate the *extent* to which assets have been safeguarded and data integrity has been maintained.

A measure of *asset safeguarding* that auditors can use is the *expected loss* that occurs if the asset is destroyed, stolen, or used for unauthorized purposes. Auditors can assign different probabilities to the different losses that could occur that is, if there is uncertainty surrounding the size of the dollar losses that result if assets are not safeguarded, the losses can be described via a probability distribution. Auditors can then calculate the expected loss if the asset is not safeguarded.

For example, assume that there is a .3 probability of losing $900,000 if an asset is destroyed (in, say, lost revenues generated via the asset), a .6 probability of losing $1,000,000, and a .1 probability of losing $1,200,000. The expected loss can then be calculated using this formula:

$$EL = \sum_i p_i L_i$$

= (900,000 • .3) + (1,000,000 • .6) + (1,200,000 • .1)
=990,000

The measure of *data integrity* that auditors use during an audit depends on their audit objectives and the nature of the data item on which they focus. Their overall concern is the extent to which a system of internal control permits errors to occur. External auditors' focus is likely to be on whether a material *dollar error* exists in the financial statements. The measure of data integrity will be the size of the dollar error that external auditors estimate exists in the accounts as a result of internal control weaknesses. Internal auditors are also likely to be concerned about dollar errors that might exist already or that might arise at some time in the future. In addition, they

might also be concerned about the existing or potential size of *quantity errors* and the existing or potential *number of errors.* For example, they might be evaluating the quantities-on-hand for inventory where their focus is whether the recorded quantities-on-hand are accurate. Similarly, they might be evaluating the accuracy and completeness of name-and-address records, where their focus is the number of records that are in error. External auditors might also be concerned about quantity errors and the number of errors, but perhaps only to the extent that they are related to dollar errors in the accounts.

If internal controls can fail stochastically, an auditor's estimate of data integrity must be made in terms of a *probability distribution of error* that might arise. For example, from time to time a clerk might enter a wrong amount at a terminal that an input validation program is unable to detect. If this error occurs randomly, the impact of data integrity will depend on factors like the nature and seriousness of the input error made, the timing of the error, and the ways it compounds or compensates with other errors that might be made. A single point estimate of the resulting error, therefore, is unlikely to suffice. Instead, auditors need to determine the shape of the probability distribution of the error that might result and the various moments of that distribution.

1.3 Nature of the global evaluation decision

When auditors make the global evaluation decision, they seek to determine the overall impact of individual control strengths and weaknesses on how well assets are safeguarded and how well data integrity is maintained. External auditors' primary focus will be on *(1)* whether material losses have occurred already because assets have not been safeguarded and *(2)* whether material errors exist in the financial statements because data integrity has not been preserved. Because external auditors will also be concerned about providing advice to management to assist them to discharge their responsibilities, they will also consider the *potential* for losses from failure to safeguard assets and to maintain data integrity. This provision of advice, however, is not their primary responsibility. On the other hand, internal auditors most likely will give equal emphasis to losses that *have occurred* and losses that *could occur* as a result of a failure to safeguard assets and to maintain data integrity. Their responsibilities are likely to be somewhat broader, therefore, than those of external auditors.

As auditors conduct tests of controls, they will continue to make global evaluation judgments. When they evaluate the reliability of each control, they will consider its impact on asset safeguarding and data integrity within the management or application subsystem or system that is their focus. To the extent that auditors believe material losses or account misstatements could have occurred, they will expand the scope and extent of their substantive testing. Likewise, as auditors conduct substantive tests, they will again be making global evaluation judgments. Once more they will expand the scope and extent of substantive testing if they conclude material losses or account misstatements could have occurred.

During the conduct of tests of controls and substantive testing, note that auditors are usually making the global evaluation judgment at the *subsystem* or *system* level. For example, they might conclude that the

planning subsystem within the head-office information systems division is defective or that the input subsystem in an order-entry system is defective. Auditors make a global assessment of the impact of these defects on the particular management system or application system that they are evaluating.

External auditors, however, must bring all these individual global evaluation judgments together. They must consider the losses or account misstatements that they estimate have occurred within individual application systems, aggregate them to the cycle level, and then aggregate them once more to the financial statements level (Figure 1-1). As external auditors undertake this aggregation process, they must consider the ways in which losses or account misstatements at the system or cycle level can compound and compensate. This final aggregation is the basis for the opinion they render on the financial statements.

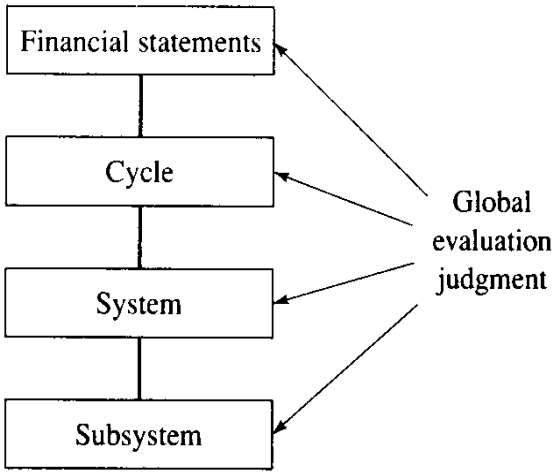


FIGURE 1-1 Levels of the global evaluation judgement

1.4 Determinants of judgment performance

Since the mid-1970s, a substantial amount of research has been undertaken to try to identify and understand the impact of those factors that impact the quality of auditor judgments. In light of the results obtained in these studies, Libby and Luft argue that the determinants of judgment performance can be usefully grouped into four categories: *(1)* the decision maker's *ability, (2)* the decision maker's *knowledge, (3)* the *environment* in which the decision maker must make his or her judgments, and *(4)* the decision maker's *motivation* level In the following subsections, we briefly examine the nature and impact of these determinants because they will help us to understand how auditors might make better judgments on how well systems safeguard assets and maintain data integrity. There are also lessons to be learned for how auditors might make judgments on how well systems achieve their effectiveness and efficiency objectives matters we will examine in the next two chapters.

1.4.1 Auditor's Ability

Whenever we seek to make a judgment about the level of asset safeguarding or data integrity maintained, we need to recognize that as humans we have various inherent cognitive strengths and limitations. Our cognitive abilities affect the quality of the judgments we will make.

For example, one of the cognitive strengths we possess as humans is our ability to identify relationships and patterns in cues that come to our attention. Consider, for example, the remarkable ways in which many people can "read the body language" emitted by another person with whom they interact. This ability to somehow assimilate diverse cues and to identify complex and abstruse patterns helps us when we must examine internal control evidence to reach a judgment on how well system safeguards assets and maintains data integrity.

The first is the *representativeness* heuristic; whereby we tend to assess the probability that one thing is representative of another thing by the degree to which the first thing resembles the second thing. For example, assume that we believe a high level of security exists in relation to a computer facility we are evaluating. For a start, this belief might or might not be well founded. Nevertheless, suppose we collect some evidence on the reliability of security controls associated with the facility. After having examined just a few controls, we find they are *representative* of reliable security controls. As humans, we then are likely to be biased toward concluding that the overall control system will be reliable, even though sampling theory might indicate that we cannot reach this conclusion on the basis of the small number of controls we have examined.

The second is the *anchoring and adjustment* heuristic, whereby we tend to start from an initial base and adjust that base to reach our final judgment. For example, during our preliminary evaluation of an internal control system, we might conclude that the controls we are examining are reliable. When we collect further evidence on the reliability of the controls, however, we might find several instances of test results that manifest that the controls are unreliable. The signals, in fact, might be clear that the control system is unreliable, but as humans we will often tend to anchor on our initial judgment that the system is reliable and under-adjust our judgment as we receive additional information.

The third is the *availability* heuristic, whereby we tend to assess the frequency of a class or the probability of some event by the ease with which we can remember instances of the class or the event. For example, we might evaluate the risk of a particular type of security exposure in an organization on the basis of the extent to which we can remember encountering the same exposure in other organizations we have audited. Unfortunately, our recall of the frequency with which the exposure occurred in other organizations might be a poor basis for assessing the risk of the exposure in the current organization we are auditing.

1.4.2 Auditor's Knowledge

Auditors' knowledge comes from two basic sources: first, the education and training received; and second, the experience accumulated in audit work

(Figure 1-2). Although the knowledge acquired from both sources overlaps to some extent, important differences also arise. For example, after you have studied this book, you should have a reasonable level of knowledge about, say, the nature of errors that could occur in data input and the controls that you could use to reduce losses from these errors. You are unlikely to have gained substantial insights, however, about the sources of inherent risk that pertain to a financial institution that undertakes complex transactions on the stock exchange. You might acquire some of this knowledge if you studied books about the stock market. Probably, however, you need to have direct experience of these institutions to have well-developed knowledge of the inherent risks associated with them.

The education and training and the experience received can also be general or specific. In the previous example, you might attend training courses that have been designed specifically to address the inherent risks associated with the type of financial institution you are auditing. You might also have carried out a large number of audits of these types of financial institution. Alternatively, most of your training might relate to general audit procedures, and most of your experience might be dispersed across a large number of different types of organizations.

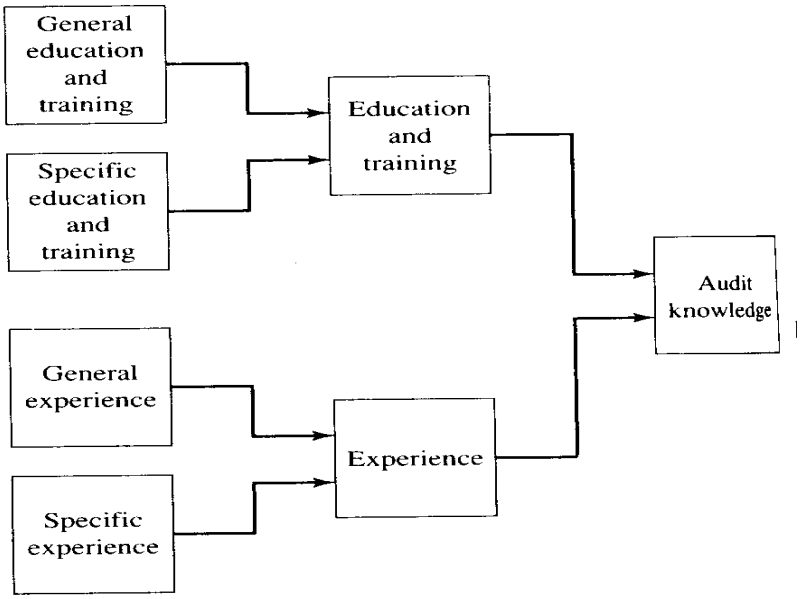


FIGURE 1-2 Source of audit knowledge

As we might expect, both general and specific knowledge play a part in determining the quality of audit judgments. For many audits, the level of gene knowledge could be an important determinant of the decision quality wt making asset safeguarding and data integrity judgments. For example, the organization we are auditing might not be complex, and it might also be representative of many other types of organizations we have encountered. As the level of specificity associated with the organization we are auditing increases, however, the complexity associated with the judgments we must make also creases. As a result, we need to have more specialized knowledge if we are make high-quality judgments.

1.4.3 Audit Environment

The audit environment describes the context in which auditors must make their evaluation judgment. Potentially, many characteristics of the environment might bear on auditor judgment processes. Libby and Luft identify four characteristics, however, that seem to be especially important.

The first is the *audit technology* that auditors can use to guide and support their audit judgment. We are using the term here in a broad sense. It includes elements like the internal control questionnaires used to structure the evaluation of internal controls, the audit standards and policies used to guide the conduct of audits, the expert systems used to help in evaluations, the way we structure audit teams to try to improve the quality of the audit, and the ways we document our findings to try to facilitate our making a final judgment. Auditors use this technology to try to overcome some of the deficiencies in their judgment processes and to extend their cognitive abilities.

The second environmental characteristic is *group processes.* One of the interesting features of the audit profession is that it uses groups extensively during the conduct of an audit; it does not rely on individual persons to make judgments. Unfortunately, research in auditing is only just beginning to provide us with some understanding of why group processes in auditing might be useful and why, therefore, auditors are likely to have incorporated group processes into their work.

The third environmental characteristic is *prior involvement* in an audit. On the one hand, prior involvement with an audit (say, in a previous year) might enhance the knowledge that auditors bring to the evaluation judgment. In this light,' it might improve the quality of their judgment. On the other hand, prior involvement might cause auditors to be less critical of the audit work they examine as it represents some of their own work. Thus, prior involvement might undermine the quality of their evaluation judgment. Again, audit research is only just beginning to provide us with insights on the effects of prior involvement on judgment quality.

The fourth environmental characteristic is *accountability.* Auditors are accountable for their work in many different ways. For example, they are accountable to their supervisors and their clients, and they will also be held accountable for their work under the law. The positive aspect of accountability is that it should motivate auditors to try to make high-quality judgments whenever they evaluate asset safeguarding and data integrity. A potential downside, however, is that auditors might become too concerned with accountability issues (like keeping extensive documentation). As a result, auditors might become exhausted and devote insufficient attention to the cognitive processes they need to exercise to make a high-quality evaluation decision.

1.4.4 Auditor's Motivation

The motivation auditors have to perform an audit task can affect *(1)* the effort they exert to perform the task and *(2)* the cognitive processes associated with the task. For example, if auditors are making a judgment about whether material errors exist in financial statements, the loss function

they face is likely to affect the amount of effort they exert to make this judgment. If a large number of people rely on the financial statements and auditors make a poor judgment, they could be sued for substantial amounts. This potential for loss is also likely to affect the *salience* of certain evidence to the evaluation judgment. For example, auditors might be more sensitive to evidence that suggests the possibility of top management fraud when many people rely on their audit opinion.

1.5 Audit technology to assist the evaluation decision

In this section, we examine some of the technology that has been developed to assist auditors to make the global evaluation decision on how well system safeguards assets and protect data integrity. By *technology,* recall from the previous discussion that we are interpreting this term in a broad sense. It is not just confined to computer technology; it also applies to any tool, structure, policy, or approach developed to facilitate undertaking the audit. Given the complexity of the evaluation judgment auditors must make, this technology can help by structuring and perhaps in part automating the evaluation decision. It also might provide auditors with insights that help them to understand the evaluation decision better.

1.5.1 Control Matrices

One of the earliest technologies developed to assist with the evaluation decision are control matrices. Control matrices can be prepared in various ways. Table 1-1 shows a common approach, however, using the example of the data capture activities associated with the input subsystem belonging to an application system.

TABLE 1-1 Input subsystem evaluation for customer order transaction class

Errors/irregularities / *Controls*	*Unauthorized customer*	*Unauthorized terms and credit*	*Incorrect quantity*	*Incorrect price*	*Untimely processing*
Order-entry operator well-trained	M	M	M	M	M
Input screen layout (quality)					
screen organization			M	M	M
field captions and entry fields			M	M	M
field alignment, justification, spacing			M	M	M
headings and messages			M	M	M
Input program					
valid customer check	H				
authorized credit		H			
inventory available			M		
Sales manager override report	M	M		L	L

Note: H = high reliability; M = moderate reliability; L = low reliability.

Note the structure of the controls matrix shown in Table 1-1. The columns of the matrix show causes of loss in this case, circumstances that would cause a loss to occur if they arose during the data capture stage of an application system's processing. The rows of the matrix are controls exercised over the causes of loss to reduce expected losses from the causes. The elements of the matrix might be some rating of the effectiveness of each control at reducing expected losses from each cause, the reliability of the control with respect to each control in light of tests of controls that have been conducted, or the marginal benefits and costs of exercising the control. In Table 1-1, the elements show a simple rating of whether a control is likely to have high, medium, or low reliability in reducing losses from a particular exposure.

To undertake the evaluation decision using the control matrix, conceptually we first examine each column of the matrix and ask the following question:

For a given cause of loss, do the controls over the cause reduce the expected loss from the cause to an acceptable level? For each cause of loss, we must somehow weigh up the effect of the various controls and determine whether the exposure that remains is at an acceptable level. We must consider the columns in total to determine whether a material loss could still arise.

A control matrix can be prepared at various levels of aggregation in our evaluation decision. In Table 1-1, we see it used at the application subsystem level. It can also be used, however, at the system level, cycle level, and overall accounts level. Basically, it is a way of bringing together some of the important factors auditors must consider when making the evaluation decision.

1.5.2 Deterministic Models

Deterministic models can be useful when evaluating part of a system of internal control or obtaining a first approximation of how well a computer system safeguards assets and maintains data integrity. For example, consider the access control mechanism in an operating system. Assume we discover an integrity flaw in the system that allows hackers, under certain conditions, to violate the privacy of a data file. In other words, the flaw can be exploited so the data file (asset) is no longer safeguarded against unauthorized use.

To determine the consequences of the flaw, we might access the system log to determine how many times the flaw has been exploited, assuming, of course, that the integrity of the log has been preserved. Calculating the loss that has resulted because of the flaw involves estimating the loss on each occasion that the flaw was exploited and summing the losses. Thus, the model used in this example is deterministic, provided there is no uncertainty about the losses involved.

Deterministic models are relatively simple to construct. Auditors might use them to perform a pencil-and-paper analysis of how well a system safeguards assets and maintains data integrity. The models provide only limited information, however, about the forms of the probability distribution of error that can be produced when a system contains stochastic elements.

In these circumstances, auditors must exercise caution when they interpret the results they obtain using them.

1.5.3 Software Reliability Models

Software reliability models use statistical techniques to estimate the likelihood that an error will occur during some time period on the basis of the patterns of past errors that have been discovered in the system. The models assume that the number of errors discovered over time increases at a decreasing rate. This assumption seems reasonable. When a system is first tested, errors are found relatively quickly. As testing proceeds, however, each additional error is harder to find. Provided that records are kept of the error discovery process, the shape of the function that relates the probability that an error will occur to the age of the system can be estimated and predictions made about the likelihood that future errors will occur. Note that the age of the system might need to be measured in terms of the number of times it has been executed rather than elapsed time. Substantial elapsed time might have occurred, but a system might be executed infrequently and after variable intervals during this period. As a result, errors might not surface because only a few execution paths through the system have been exercised.

1.5.4 Engineering Reliability Models

Engineering reliability models allow auditors to estimate the overall reliability of a system as a function of the reliability of the individual components and individual internal controls that make up the system. In this respect they mirror the auditor's traditional approach to assessing the overall reliability of a system based on evidence collected about the reliability of individual internal controls within the system. In some ways the assumptions underlying the models are restrictive. Moreover, their practical usefulness is still questionable. Nevertheless, the models provide important insights into how control strengths and weaknesses can compensate and compound to affect the overall reliability of a system. As such, they help auditors to understand how the overall evaluation of internal control should be made.

1.5.5 Simulation Models

Whenever possible, auditors should use analytical models toTielp them make the evaluation judgment. With analytical models, usually auditors can evaluate the value of the dependent variables that interest them at low cost over a wide range of variations in the independent variables and the structure of the model. The deterministic models and probabilistic models examined previously, for example, often can be developed and solved quickly and cheaply.

Sometimes, however, analytical models are not appropriate to use when auditors make the global evaluation judgment. For a start, they might conclude that the assumptions underlying the models are too restrictive and thus do not mirror reality sufficiently well. In addition, analytical models might not be mathematically tractable because the appropriate equations to use to model the system are not obvious or the equations appear to be

insolvable. In these cases, auditors might consider using simulation models to assist their making the global evaluation judgment.

When auditors use a simulation model, they can evaluate the behavior of a system over time. For example, consider how we might use a simulation model to evaluate asset safeguarding. We might be evaluating a security system in which there is some probability that a guard will fail to detect an intruder, some probability that a surveillance system also might fail to detect the intruder, some probability that the intruder can crack a password system, and so on, and eventually the intruder gains access to and steals sensitive data files. Often an analytical model can be used to determine the probability of all controls failing. Nevertheless, if controls compound and compensate in different ways, we might need to use a simulation model to obtain the needed insights for us to he confident in our evaluation of the reliability of the system of controls. Moreover, we can study how the security system responds over time as it is subjected to various types of threats. We are not basing our decision, therefore, on a one-off situation in which the security system might or might not detect the intruder.

1.6 Cost-effectiveness considerations

So far our discussion has proceeded without our considering the costs of safeguarding assets or maintaining data integrity. Reliable systems can be achieved, however, only at a cost. Whenever we invest in making a system more reliable, we must consider whether the benefits we expect to obtain exceed the costs we expect we will incur.

In the following subsections, we briefly examine some aspects of the judgment auditors must make on whether the controls in place are cost-effective. We deal with this decision only at a reasonably intuitive level. Some of the approaches that have been developed to help us to make this decision, however, are somewhat complex.

1.6.1 Costs and Benefits of Controls

Implementing and operating controls in a system involves five costs:

1. Initial setup costs must be incurred to design and implement controls. For example, a security specialist might have to be employed to design a physical security system, and a systems analyst might be assigned the task of designing the validation routines for an input program. When these design tasks are completed, costs will then be incurred, for example, to install magnetic card door locks and to write programs.

2. Costs associated with executing controls will be incurred. For example, the wages of a security officer must be paid, and the costs associated with using a processor to execute input validation routines must be met.

3. When a control signals that some type of error or irregularity has occurred, costs will be incurred to search for the error or irregularity, to determine whether one exists (that is, the control has operated reliably in signaling an error or irregularity), and to correct any errors or irregularities that are found.

4. Costs arise because controls do not detect some errors or irregularities (the control might malfunction or it might not have been designed to detect the error or irregularity that has occurred), and costs arise because the control system fails to correct errors and irregularities properly when they are discovered. These undetected or uncorrected errors or irregularities then cause losses. For example, an uncorrected error or irregularity could allow a defalcation to occur.

5. Maintenance costs are incurred to ensure the controls are kept in correct working order. For example, periodically a security guard must be retrained, and input validation routines might have to be rewritten as the format of input data changes.

The first cost described here is the outlay for a system of controls. The remaining four costs are the ongoing operational costs of a control system.

1.6.2 A Controls Matrix View of the Cost-Effectiveness of Controls

Earlier in the chapter we examined the use of control matrices to evaluate whether a set of controls had reduced exposures to an acceptable level. Control matrices can also be used to help us to characterize the decision on whether controls are cost-effective.

When we examine a column of a control matrix, recall that we are seeking to determine whether a set of controls reduces an exposure to an acceptable level. In essence, we are asking whether the reduction in expected losses exceeds the costs of designing, implementing, operating, and maintaining the set of controls. To determine whether we should put additional controls in place, we also need to consider whether the marginal reduction in expected losses from .the exposure from having additional controls will exceed the marginal costs of the controls.

Even if a control is not cost-effective in terms of a *single* exposure, however, it might be cost-effective in terms of *all* the exposures where it acts to reduce expected losses. In this light, we must also consider each *row* of the controls matrix. As we work across each of the rows in Table 1-1, for example, we are considering the reduction in expected losses that occurs for each of the exposures listed in the columns with regard to the particular control in the row we are considering. In other words, we focus on a particular control by choosing a row in the controls matrix, and we consider the impact of this control on all exposures by working across the row in the controls matrix.

The *global* evaluation question involves our asking this question: What is the optimal set of controls for the organization? The answer to this question somehow involves our undertaking a joint evaluation of the columns and

rows in the controls matrix. Whereas from a columnar perspective, it might not be worthwhile to have a control, from a row perspective the benefits of the control when it is exercised over *all* exposures might exceed its cost.

Unfortunately, there are two complicating factors when auditors seek to make this global evaluation decision. First, as discussed previously, the marginal benefits and costs of exercising a control might depend on what controls are in place already and the reliability of these controls. In other words, the benefits and costs of a control are *conditional* parameters. Second, there is an overriding constraint on how many controls should exist in a system. This constraint applies when for all controls that still might be exercised the marginal benefits of any one control exceed the marginal costs of that control. In short, although auditors might understand conceptually the nature of the global evaluation decision on controls, the pragmatics associated with making this decision are difficult.

1.6.3 Controls as an Investment Decision

The design, implementation, operation, and maintenance of a control produce a stream of benefits and costs over its life. As discussed previously, at the outset, costs are incurred associated with designing and implementing the control. Each year, benefits are then obtained in the form of reduced expected losses from exposures. Each year, costs are also incurred associated with operating and maintaining the control. In this light, we should conceive of a control as a form of investment. At least conceptually, we should calculate the net present value for each control and invest in the control if its net present value is greater than or equal to zero. Where there are competing controls (controls that reduce expected losses for the same exposures), we should invest in that control which has the highest net present value. Because the costs of evaluating each control we might implement are likely to be excessive, we will probably need to focus our evaluation on a *set* or *system* of controls. In other words, we will consider the set or system of internal controls as the investment rather than individual controls as the investment.

One difficulty we will face in considering controls as an investment is to estimate the size of the stream of benefits and costs that will occur during each period of the control system's life. Perhaps the more difficult decision we will have to make, however, is to determine the appropriate discount rate to use in our net present value calculations. Current finance theory tells us that the discount rate, *k,* that we should use should be calculated as follows:

$$k = k_f + \beta(\overline{K_m} - k_f)$$

where:

k_f = risk-free rate of return

$\overline{K_m}$= expected rate of return on the market portfolio

β = beta coefficient of a security

The β coefficient of a security indicates the riskiness of returns on the security relative to returns on the market portfolio. The value of *p* can be obtained by regressing the returns on the security to returns on the market portfolio.

What is the β we should use, however, for an investment in a control system? Conceptually we need to find a firm that operates in the market that invests only in the control system we are seeking to evaluate. Of course, practically we know that such firms are unlikely to exist. In this light, we might use the β of the firm that is seeking to design, implement, operate, and maintain the control system we are evaluating and make some adjustment to β (upwards or downwards) depending on our estimate of the risk associated with the control system relative to the overall risk associated with the firm.

2. Evaluating System Effectiveness

Structure

Objectives

After going through this unit, you should be able to:

- understand to overview of the effectiveness evaluation process
- understand to A model of information system effectiveness
- discuss for varies evaluating system techniques

2.1 Introduction

In this lesson, we focus primarily on the first objective of a post-implementation review namely, evaluating systems to determine how well they meet their objectives. The first section will provide an overview of how we undertake an evaluation of system effectiveness. Next, we examine a model that has been developed to show the major factors that are believed to affect information system effectiveness. We then discuss the influence of each of these factors on information system effectiveness. Our goal is to develop an understanding of the nature of each of these factors, the ways they are interrelated, and the approaches we might use to measure them and their impact.

2.2 overview of the effectiveness evaluation process

Ideally, all information systems should be subjected periodically to a post implementation review to assess how well they are meeting their objectives. Empirical research has shown, however, that only certain systems undergo post implementation evaluations. Several factors appear to affect which information systems are selected for review. For example, if top managers have few doubts about the success of a system, they might not request a post implementation review. Conversely, if they have substantial doubts about the success of the system, they may commission a review.

Reviews also can be undertaken for political reasons. For example, managers might request that a review be undertaken on a system for which they are responsible even when they know the system is a success. By having someone independent confirm its success, they might be seeking to enhance their standing with senior management. They might have an expectation that they can then extract more resources from the organization for their own purposes. Whenever auditors are requested to undertake a post implementation review, therefore, they should be circumspect about the underlying motivations for the evaluation.

An evaluation of system effectiveness involves six steps:

1. *Identify the objectives of the information system.* Sometimes the objectives might have been stated clearly when the system was first developed. Sometimes, however, the objectives might be vague and ill defined. Different stakeholders in the information system also might have different objectives for the system. Somehow auditors must tease out the objectives that each stakeholder group has for the system so they can determine which of these objectives have been achieved.

2. *Select the measures to be used.* Auditors need to he able to measure the extent to which each objective they identify for the system has been achieved. In some cases, they might use quantitative measures that they obtain via, say, questionnaires administered to users or statistics relating to productivity. In other cases, auditors might use qualitative measures obtained via. Say, interviews with and observations of users.

3. *Identify data sources.* Having chosen the measures they wish to obtain, auditors must then identify the best sources of data for these measures. In some cases, it might be various types of users. In other cases, it might he, for examples, manufacturing records on productivity, wastage, spoilage, and so on that are maintained routinely by an organization.

4. *Obtain* ex ante *values for measures.* When auditors have identified the measures and the best sources of data for these measures, they must attempt to determine the values of these measures before the system they are evaluating was implemented. Auditors need a basis for establishing the impact of the system. Unless these *ex ante* values were collected prior to the implementation of the system, it could be difficult to obtain them after the system is operational.

5. *Obtain* ex post *values for measures.* After the system is implemented, auditors then must collect data on the measures they have chosen to evaluate effectiveness. One difficulty they face is determining what time period should elapse before the measures should be taken. It might take some time before the effects of an information system on an organization begin to stabilize. It might also be important to collect data on these measures over time if they are interested in the patterns of changes that are occurring.

6. *Assess the system impact.* When auditors have values for the *ex ante* and *ex post* measures, they can then assess the impact of the system by comparing the values for the two sets of measures. It is important that they try to look beyond the measures to understand the reasons for any changes they observe. Their report will be more useful to management if they can account for the changes they have identified.

2.3 A model of information system effectiveness

For many years, researchers in the information systems discipline have attempted to understand what we mean by information system 'effectiveness, to develop valid and reliable measures of information system effectiveness, and to identify the major factors that affect information system effectiveness. Unfortunately, these goals have proved especially difficult to achieve. Although we have made some progress, much research is still needed if we are ever likely to obtain a thorough understanding of the nature of information system effectiveness and the factors that impact information system effectiveness.

Figure 2-1 shows a model of information system effectiveness that is intended to be an amalgam of work that has been carried out by several researchers. We examine each of the components of this model in the following sections. In summary, however, the model manifests a set of hypothesized relationships among factors that are thought to have an impact ultimately on whether an information system is effective. First, the quality of the system and the quality of the information it produces are hy-

pothesized to affect whether users perceive the system to be both useful and easy to use. These two perceptions are also affected, however, by users' beliefs about their abilities to use computers competently (self-efficacy). Users' perceptions about the usefulness and ease of use of the system in turn affect how they use the system for example, the frequency with which they use the system and the ways in which they use the system. How they use the system then affects their performance in their organizational role and ultimately the overall performance of the organization. How they use the system also affects their satisfaction with the system. There is also a hypothesized two-way relationship between satisfaction and individual impact. To the extent users are more satisfied with a system, it is likely to have a greater effect on them. Similarly, to the extent the system has a greater positive (negative) effect on them, they are more (less) likely to be satisfied with the system

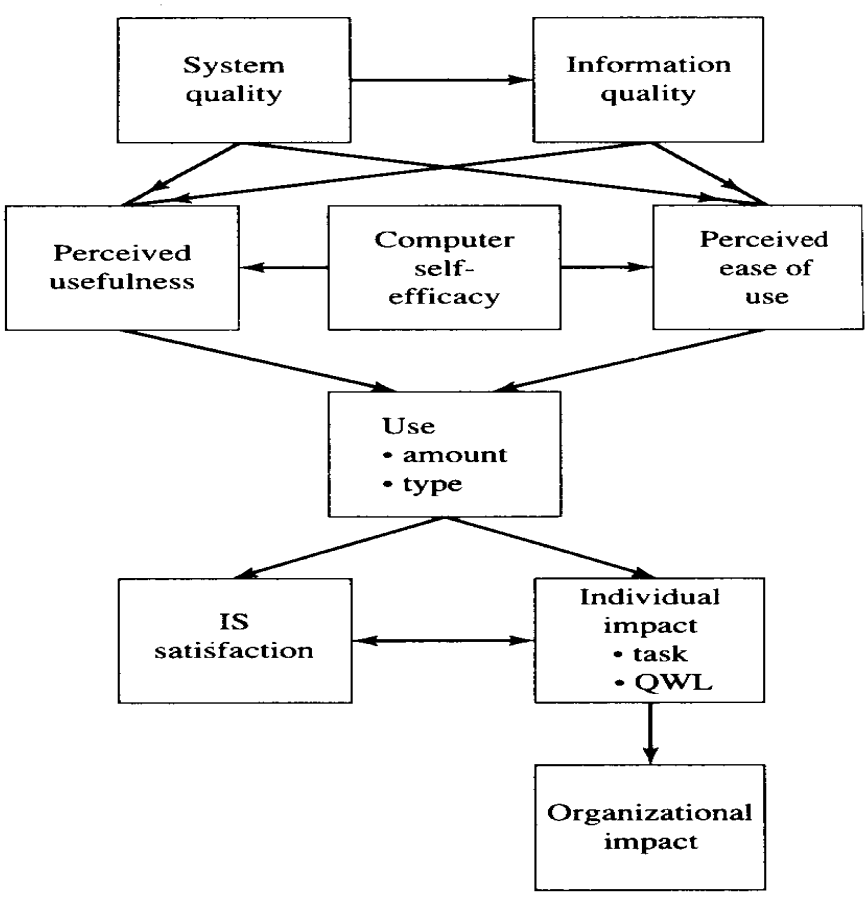


FIGURE 2-1 A model of factors affecting IS effectiveness

Auditors can use the model shown in Figure 2-1 in two ways. First, they can use it to structure their approach to the collection of the data they will need to make a judgment about whether a system meets its objectives effectively. Each component in the model indicates the types of evidence auditors must collect to be able to reach a global evaluation about the effectiveness of the system. Second, auditors can use the model to help

them understand why a system might be effective or ineffective. In some ways they could approach the evaluation of system effectiveness by simply collecting evidence about the individual and organizational impacts that arise as a result of implementing and operating a system the final components in the model. If auditors do not collect data on the preceding components, however, they will not have a sound basis for determining why a system is effective or ineffective. For example, if auditors find a system has not had the desired individual and organizational impacts, management might ask them to provide the reasons why the desired impacts have not occurred and the remedial actions they should take. Auditors can use the model to help them think about potential causes and to provide a basis for their making recommendations to try to improve the effectiveness of the system.

2.4 Evaluating system quality

Potentially many characteristics of the hardware and software components of an information system might affect users' perceptions of the usefulness and ease of use of the system. One set of characteristics will be fairly apparent to users after they have interacted with the system for only a short period of time. It includes the following:

1. Response time (online system),
2. Turnaround time (batch system),
3. Reliability (stability) of the system,
4. Ease of interaction with the system,
5. Usefulness of the functionality provided by the system,
6. Ease of learning,
7. Quality of documentation and help facilities, and
8. Extent of integration with other systems.

From the viewpoints of the users of a system, however, these factors tend to have a somewhat immediate impact on their attitudes toward the system. If users find it difficult to interact with a system, for example, they are unlikely to have favorable perceptions about the usefulness of the system and the ease with which they can use the system to accomplish their goals.

Some of these factors are also associated with software effectiveness. To assist auditors to make a judgment about software effectiveness, they should examine four attributes of the software used to support the system:

1. *History of repair maintenance.* The history of program repair maintenance indicates the quality of a program's logic. Recall that repair maintenance is carried out to correct logic errors. Extensive repair maintenance means inappropriate design, coding, or testing technologies have been used to implement the program.

2. *History of adaptive maintenance.* Adaptive maintenance is carried out to alter a program to accommodate changing requirements. There are two reasons why it occurs. First, program designers might have

formulated incorrect specifications in the first place. As a result, the specifications have to be changed and the program logic altered. Incorrect program specifications mean they should examine the approaches used to develop the specifications. Second, user requirements might change. As a result, the program has to be altered to meet these new user requirements. Nonetheless, frequent modifications to meet changes in user needs might mean the program is inflexible; in other words, it has not been designed to accommodate change.

3. *History of perfective maintenance.* Perfective maintenance is carried out to improve program resource consumption that is, to make the program execute more efficiently. Large amounts of perfective maintenance might mean that the program has been designed poorly. Alternatively, perhaps the hardware platform on which the program runs is being changed frequently and the program has to be constantly tuned to adapt to the new platform.

4. *Run-time resource consumption.* If at run time an application program consumes resources inefficiently, it could mean that it is poorly designed or that the programming language or compiler used is inappropriate for the task to be performed. Auditors should examine the technology used to support these aspects of software implementation.

The designers and programmers responsible for carrying out program modification and repair maintenance and the operators responsible for running programs can provide auditors with information on the appropriateness of the software technology used to support an application system. Designers and programmers can make judgments on the overall quality of programs. For example, they know whether a program is easy to modify or repair. Operators often can make judgments on whether a program consumes abnormal amounts of resources at run time.

2.5 Evaluating information quality

The quality of the information produced by an information system can have important effects on users' perceptions of the usefulness of the system and ease of use of the system. Some of the attributes of information quality that auditors might seek to measure are the following:

1. Authenticity,
2. Accuracy,
3. Completeness,
4. Uniqueness (nonredundancy),
5. Timeliness,

6. Relevance,

7. Comprehensibility,

8. Precision,

9. Conciseness, and

10. Informativeness.

When auditors evaluate the quality of information produced by a system, note that they are basically trying to assess how well the information enables users to undertake their jobs. Nonetheless, auditors should be sensitive to the fact that the information provided by a system can have other impacts on users' lives. For example, users might have a negative view of the quality of information provided by a system if they perceive that the information is used by management to gain increased power over them.

2.6 Evaluating perceived usefulness

Several items have been used to measure the perceived usefulness of an information system. Davis has found the following items to be valid and reliable, however:

1. Users perceive that the information system enables them to accomplish tasks associated with their job more quickly.

2. Users perceive that the information system enables them to improve their job performance.

3. Users perceive that the information system enables them to increase their productivity.

4. Users perceive that the information system enables them to increase their effectiveness on the job.

5. Users perceive that the information system makes it easier for them to undertake the tasks associated with their job.

6. Users perceive the information system to be useful in their job.

If auditors are seeking to evaluate the perceived usefulness of an information system, therefore, they might wish to use these items in a questionnaire or as the basis for any interviews they conduct with users.

2.7 Evaluating perceived ease of use

Several items have been used to measure the perceived ease of use of an information system. Davis has found the following items to be valid and reliable, however:

1. Users perceive that it is easy for them to learn to operate the information system.

2. Users perceive that it is easy for them to get the information system to do what they want it to do.

3. Users perceive that they can interact with the system in a clear and understandable way.

4. Users perceive that interaction with the information system is flexible.

5. Users perceive that they can quickly become skilful with the information system.

6. Users perceive that the information system will be easy to use.

If auditors are seeking to evaluate the perceived ease of use of an information system, as with perceived usefulness they can use these items in a questionnaire or as the basis for any interviews they conduct with users. Moreover, as with perceived usefulness, auditors should be mindful of other factors (like quality-of-working-life effects) that might affect users' perceptions of the ease with which an information system can be used.

2.8 Evaluating computer self-efficacy

Several researchers believe that computer self-efficacy is an important variable in accounting for the likely effectiveness of an information system. In this regard, Figure 2-1 shows that computer self-efficacy is predicted to affect users' perceptions of the usefulness of and ease of use of an information system. If users have a poor perception of themselves in terms of their competence to use computers, it is unlikely that they will perceive the output provided by a system to be useful or that the system will be easy to use.

If auditors wish to measure users' computer self-efficacy, they will need some kind of measurement instrument. In this regard, Compeau and Higgins have developed a questionnaire designed to measure computer self-efficacy. Initial research conducted using the questionnaire indicates that it is valid and reliable. It first asks respondents to consider a new software package designed to assist them with their work. Under various conditions, it then asks them whether they think they could complete their job using the software package and, if they answer yes, how confident they feel in their judgment. Examples of conditions are no one being available to tell users what to do, only the software manuals being available for reference, and the respondent having used a similar package before.

In some cases, computer self-efficacy might not be a major concern in terms of whether an information system is likely to be effective. For example, auditors might be dealing with users who have extensive experience with similar sorts of computer systems to the one they are evaluating. Where such conditions do not exist, however, auditors should be mindful of the impact that users' perceptions of computer self-efficacy might have on the effectiveness of an information system.

2.9 Evaluating information system use

If people perceive that an information system is useful and that it is easy to use, research indicates that they will have positive attitudes toward the system. These positive attitudes in turn will translate into favorable

intentions toward using the system. Intentions have been shown to be a good predictor of actual use of the system.

In the interests of simplicity, however, Figure 2-1 does not show the link between perceived usefulness and perceived ease of use of an information system and a user's attitude toward the system, nor does it show the link between attitude and behavioral intention. Rather, we focus instead on how the information system is used. In this regard, the concept of information system usage turns out to be problematical. In the following subsections, we briefly explore various notions of usage.

2.9.1 Voluntary versus Involuntary Use

Whenever auditors measure information system use, they must be careful to determine whether use is voluntary or involuntary. In some systems, users can choose whether they employ a system to help them with the tasks they are performing. In other systems, reports are generated on a routine basis, and users receive the reports unsolicited. Users might also be compelled by management to use the output provided by the information system.

If system use is voluntary, auditors can build monitors into the system to determine unobtrusively how often the system is invoked by users to perform different tasks. If use is involuntary, however, auditors must then attempt to determine whether use is "real" or "apparent." Auditors can try to obtain evidence, for example, on whether system output has actually been employed to undertake a task. Alternatively, they might use interviews or questionnaires to gauge employees' real use of the system. Auditors should take care, however, if they use interviews or questionnaires to obtain data about system use from employees. The results they obtain might not be reliable. Users might not accurately recall how frequently they use a system, for example, or they might overstate frequency of use to gain favor with management.

2.9.2 Amount and Frequency of Use

Auditors can employ amount of use and frequency of use of an information system as a means of trying to establish "how much" the system is used. There are various measures of amount of use that might be helpful. For example, auditors could use the following:

1. Duration of connect time to the system,

2. Number of inquiries made,

3. Number of functions invoked in the system,

4. Number of records accessed in database,

5. Number of reports generated, and

6. Size of chargeout costs for system use.

Problems exist with all these measures, however, in terms of establishing the impact of an information system on users. For example, system connect time might be high, but the system might be idle for much of the time that the connection exists. Likewise, connect time might be high because users have little competence in using the system. As a result, they consume excessive resources to produce the output they require.

In light of these problems, frequency of use is sometimes employed as a measure of system use. Frequency of use might be a good indicator of a user's reliance on an information system. If users frequently invoke a system's functionality, presumably they find the system useful in performing their tasks. Indeed, the amount of use (as measured by, say, connect time) might be low because users can quickly accomplish their tasks. Fast but frequent access of the system, however, might still indicate that the system is effective.

2.9.3 Nature of Use

Ginzberg has argued that frequency of use is often an inappropriate measure to evaluate system effectiveness. He points to situations in which a system is used infrequently yet it is considered to be successful. For example, the act of building a decision support system could provide important insights into how an ongoing problem should be approached. In this light, users might consider the system to be successful even though their day-to-day use of the system may be low.

As a consequence, Ginzberg argues that the overall success of a system must be evaluated in terms of the *way* if is used and not just the frequency of use. The way a system is used evokes different types of change in a user's actions.

Auditors should be careful, therefore, not to conclude that a system has high operational effectiveness solely on the basis of its amount or frequency of use. Ginzberg's research suggests that the level of change brought about by the system also must be examined. In this light, auditors also need to examine the way in which a system is being used to assess its effectiveness.

2.9.4 Source of Use

Auditors might also need to determine *who* uses an information system when they assess its effectiveness. In some cases, users themselves might interact directly with the information system. In other cases, an intermediary might act on their behalf. For example, for some decision support systems that have been developed to facilitate group work, a person called a "chauf-feur" is often needed to assist users to work with the system. The chauffeur is an expert in the group decision support system who helps users to "drive" the system. If users interact infrequently with the system, they might need assistance from someone who is familiar with the system to deal with its complexities.

2.10 Evaluating individual impact

The impact of an information system on users can be manifested in several ways. Two major types of impacts that auditors need to consider when they assess a system's effectiveness, however, are task accomplishment impacts and quality-of-working-life impacts. In particular, auditors need to remember that an interaction most likely will exist between the two. If a system improves users' task accomplishment, it might also improve their quality of working life. ,For example, they might be more satisfied with their jobs. Similarly, if a system improves users' quality of working life, it may also improve their task accomplishment. For example, if a system allows users greater opportunities to use their abilities on the job, it might improve their task accomplishment. On the other hand, if a system produces a negative impact on one, the interaction effects might produce a negative impact on the other. In the following subsections, therefore, we will seek to obtain a better understanding of these two types of outcome.

2.10.1 Task Accomplishment Impacts

An effective information system improves the task accomplishment of its users. The following are some general measures that auditors can use to try to determine whether a user's task accomplishment has improved:

1. Decision accuracy,
2. Time to make decision,
3. Decision confidence,
4. Effectiveness of decision,
5. Quality of product or service produced,
6. Customer satisfaction with product or service produced, and
7. Time to undertake task.

Often, however, auditors need to identify specific measures of task accomplishment to determine whether an information system is effective. Unfortunately, performance measures for task accomplishment differ considerably across applications (and sometimes across organizations).

To illustrate this problem, consider the ways task accomplishment might be assessed for a manufacturing control system, a sales system, and a welfare system that supports counselors in their work. Some measures of task accomplishment that auditor might use for the manufacturing control system follow:

1. Number of units output,
2. Number of defective units reworked,
3. Number of units scrapped,
4. Amount of waste produced,

5. Amount of downtime, and
6. Amount of idle time.

For the sales system, some measures of task accomplishment that auditor might use follow:

1. Dollar value of sales made,
2. Changes in customer satisfaction ratings,
3. Amount of doubtful/bad debts that arise,
4. Average time for delivery of goods to customer,
5. Number of new customers acquired, and
6. Number of sales made to old customers.

For the welfare system, some of the measures of task accomplishment that' auditors might use follow:

1. Number of clients successfully counseled,
2. Average cost per client,
3. Number of clients returning for counseling, and
4. Client satisfaction ratings of counseling service provided.

2.10.2 Quality of Working Life Impacts

Besides affecting task accomplishment, an information system can also affect its users' quality of working life. Auditors must be mindful of this impact, because important relationships appear to exist between the quality of working life of people and their physical and mental health. For example, research has found associations between the characteristics of work and the incidence of heart disease, peptic ulcers, arthritis, psychosomatic illness, alienation, and suicide among employees. In many countries, the direct and indirect costs of employee ill health are substantial.

What are the factors that contribute to a high quality of working life? The following are often listed:

Quality of Work Life Attribute	*Explanation*
Adequate and fair compensation	The income received from work should meet social standards of sufficiency and bear an appropriate relationship to the income received from other work.
Safe and healthy working conditions	The physical work conditions should minimize the risk of illness and injury. There should be limitations on hours worked.
Opportunity to use and develop human capacities	Jobs should provide autonomy, involve both planning and implementation activities, allow use of multiple skills, and be meaningful. Employees should obtain feedback on their actions.
Opportunity for continued growth and security	There must be ongoing opportunities to develop and use new skills. Employment and income security should exist.
Social integration in the work organization	The workplace should be free of prejudice, allow interpersonal openness, and encourage a sense of community.
Constitution in the work organization	The workplace should preserve personal privacy, allow free speech, provide equitable treatment, and allow due process when disputes arise.
Balanced work role and total life space	Work should be integrated with the total life space; e.g., it should not place unreasonable demands on leisure and family time.
Social relevance of work life	Employees should perceive the workplace to be socially responsible.

If auditors try to use these factors to assess the impact of an information system on the quality of working life of its users, they will encounter two problems. First, they will find that different users have different perceptions of what constitutes a high quality of working life. For example, some will consider the quality of working life from a productivity perspective, some from physical conditions and wages perspective, and some from an alienation perspective. In this light, auditors should recognize the limitations of overall measures of the quality of working life as opposed to measures tailored specifically for individual persons.

Second, it is often difficult to find valid and reliable measurement instruments to assess the quality of working life. One problem that the designers of these instruments face is that somehow their measures must take into account the variation in responses by persons to the same environment. For example, a person who has had several jobs is likely to have a higher level of satisfaction with a particular job than a person who is employed for the first time. The time span for measurement also must be chosen. Employees who are subject to poor working conditions might report a high quality of working life if they have high expectations of better things to come. The measures also must be verifiable and not subject to

manipulation. Otherwise, responses could be biased intentionally by a particular person or interest group to further their cause.

Because of these problems, one approach auditors can adopt to assess the quality of working life is to use surrogate measures that is, measures that act as indicators of the level of the quality of work life existing instead of dir measuring attributes of the quality of working life. Some surrogate measures that have been widely used follow:

$$\text{Absenteeism rate} = \frac{\text{total absent days}}{\text{total working days}}$$

$$\text{Tardiness rate} = \frac{\text{total incidents of tardiness}}{\text{total working days}}$$

$$\text{Strike rate} = \frac{\text{total strike days}}{\text{total working days}}$$

$$\text{Work ban rate} = \frac{\text{total work bans}}{\text{total working days}}$$

$$\text{Stoppage rate} = \frac{\text{total stoppages}}{\text{total working days}}$$

$$\text{Grievance rate} = \frac{\text{total grievances}}{\text{average work force size}}$$

$$\text{Turnover rate} = \frac{\text{total turnover incidents}}{\text{average work force size}}$$

$$\text{Accident rate} = \frac{\text{total accidents}}{\text{total working days}}$$

$$\text{Sick rate} = \frac{\text{total sick days}}{\text{total working days}}$$

$$\text{Theft/sabotage rate} = \frac{\text{total theft/sabotage incidents}}{\text{average work force size}}$$

Changes in these measures manifest changes in the quality of working life. For example, a lowered quality of working life can cause increased turnover of employees or a greater number of strikes. Thus, to assess the effectiveness of an information system, the focus is on how these measures change after the system has been implemented.

2.11 Evaluating information system satisfaction

Several instruments to measure information system satisfaction have been developed that auditors might use during an audit to evaluate the effectiveness of an information system. Some examples of the types of items included in these instruments are the following:

1. Relationships with information system staff,
2. Processing of system change requests,
3. Timeliness of information,
4. Level of information system training provided to users,

5. Relevance of output,

6. Amount of output,

7. Quality of documentation provided, and

8. Dependability of the information system.

The users of an information system are then asked to rate how satisfied or dissatisfied they are with these items.

Whenever auditors use an instrument to measure information system satisfaction, however, they need to consider whether they should tailor the instrument to the specific type of information system they are evaluating. Instruments that have been developed for a batch general ledger system, for example, might not provide valid and reliable measures of user satisfaction with a decision support system. Although questions about user satisfaction with the interactive features of the latter system are' likely to be important, these same questions most likely will be irrelevant in the context of user satisfaction with the former system.

Auditors should note, also, that the distinction between information system satisfaction and other measures like perceived usefulness and perceived ease of use of an information system is not clear-cut. Some of the items included in published instruments for measuring information system satisfaction are similar to those included in instruments to measure perceived usefulness and perceived ease of use of an information system. When auditors evaluate the effectiveness of an information system, therefore, they need to be circumspect about the potential overlap between measures.

2.12 Evaluation organizational impact

If an information system has a positive impact on the people who use it, presumably it will also have a positive impact on the organization in which these people are members. The relationship between the impact on people and the impact on their organization, however, is not straightforward. People might become more efficient in performing their jobs, for example, but they still might not contribute significantly to the attainment of the organization's overall goals.

In the following subsections we examine the potential impact of an information system on an organization from two perspectives: first, from the viewpoint of its impact on the overall effectiveness of the organization: and second, from the viewpoint of its impact on economic effectiveness. The latter impact is simply one aspect of the former impact. However, often auditors will be asked to give particular attention to the economic impacts of an information system. Moreover, some difficult issues arise when evaluating the economic impact of an information system.

2.12.1 Organizational Effectiveness

What are (should be) the goals of a high-quality information system? This question turns out to be a frustrating and difficult one to answer. One possible response is that the overall objective of an information system is to

increase the effectiveness of the organization it services. This response simply shifts the problem; however, for the next question we must ask is this: What are the goals of an effective organization? The goals of an information system and the goals of the organization it serves are inextricably intertwined. Information systems are developed to help an organization meet its goals. Thus, whether or not a system is effective must be assessed in terms of organizational goals.

Unfortunately, little consensus exists on what constitute the goals of an organization. When Steers undertook a review of the literature on organizational effectiveness, for example, he found that many different types of indicators had been used to measure goal accomplishment. They included measures of profitability, growth, turnover, absenteeism, job satisfaction, stability, flexibility, morale, and readiness. We might debate whether some of these indicators measure goal accomplishment or the state of factors that affect .goal accomplishment. For example, economists might argue that ultimately the effectiveness of organizations is solely a function of their profitability. Factors like turnover, stability, readiness, and absenteeism all affect profitability. An organizational theorist might argue, however, that profitability is too gross a measure of effectiveness for it to be especially useful. Managers need to understand how well their organizations are performing on the factors that ultimately could affect profitability, such as the ability of the organization to adapt to changes in its environment and the quality of working life of its employees.

An important model of organizational effectiveness, however, has been proposed by Quinn and Rohrbaugh and Quinn and Cameron. This model, called the *competing values model,* has two dimensions of organizational effectiveness: a focus dimension and a structure dimension (Figure 2-2). In terms *of focus,* organizations can pursue goals that address either *external concerns,* such as pressure from shareholders and environmental lobbyists, or *internal concerns,* such as the well-being of employees and the level of morale within the organization. In terms of *structure,* organizations can pursue goals that address *either flexibility concerns,* such as environmental monitoring and maintaining readiness for change, or *stability concerns,* such as maintaining control over operations.-

The competing values model underscores the difficulties that exist in terms of how organizations often have to pursue different goals to survive. Somehow, management must balance these goals so that pursuit of one does not substantially undermine another to the point where the organization is unable to survive. Indeed, Cameron (1986) argues that an important characteristic of effective organizations is the way they manage paradox. Somehow they achieve goals concurrently that are in conflict with one another for example, high specialization of roles to achieve efficiencies and at the same time high generalization of roles to achieve flexibility. Moreover, the emphasis that management needs to give to one set of goals versus another set might vary throughout the life of an organization. For example, a young organization might place more emphasis on flexibility and innovation and less emphasis on profitability. A mature organization, on the other hand, might place more emphasis on profitability and control and less emphasis on human-resource issues.

When auditors evaluate the effectiveness of an information system, therefore, they need to frame their evaluation in the context of the goals that the stakeholders in the information system are seeking to pursue. It will not be helpful, for example, if auditors evaluate an information system on the basis of how much it has contributed to productivity when its stakeholders' goals are more oriented toward promoting flexibility and openness within an organization. Such an evaluation could be misleading, if not useless.

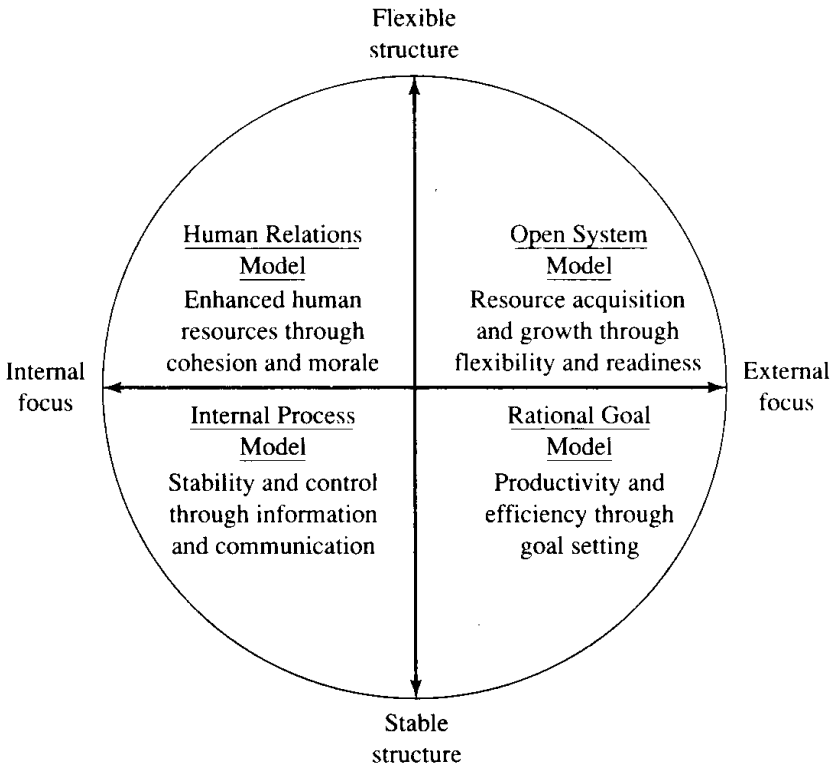


Figure 2-2 Competing values model of organizational effectiveness

Auditors need to recognize, also, that the goals held by stakeholders in relation to an information system may be both overt and covert. In other words, they might be stated formally or at least discussed openly and perhaps agreed upon by the stakeholders in the system. Alternatively, they might not be articulated formally or discussed openly, perhaps because of the political damage that might accrue. For example, management might have a covert goal to gain more control over employees, which might cause major difficulties with a union if it were to be articulated openly.

The importance of different goals among different stakeholder could also vary, and in some case the goals might even conflict with one another. For example, a group of donors to a charitable organization might view the effectiveness of an information system in terms of how well it permits efficient use of scarce funds. The professionals who work within the charitable organization, however, might be more concerned with how well the information system permits them to provide services to their clients.

In summary, whenever auditors evaluate the impact of an information system on an organization, they have to take great care at the outset to reach a good understanding of the important stakeholders in the information system and the goals they have for the system (both overt and covert). If auditors do not have a sound understanding of the stakeholders and their goals, the usefulness of their effectiveness evaluation will be undermined. Auditors need to recognize that they could have to evaluate an information system on multiple bases and that sometimes these bases might be incongruous with one another. Auditors also need to be careful whenever they make judgments about the merits of one set of goals over another set of goals. These judgments might reflect that they have adopted the perspective of one group of stakeholders only and that they are not "seeing" the system from the viewpoints of other groups of stakeholders.

2.12.2 Economic Effectiveness

One particular type of effectiveness that management is likely to ask auditors to evaluate in relation to an information system is economic effectiveness. In particular, management might be concerned with whether an information system has contributed to the profitability of an organization.

Evaluating the economic effectiveness of information systems has proved to be an especially difficult task. During the late 1980s and early 1990s, for example, several researchers pointed to a phenomenon that they called the "productivity paradox". They observed that organizations were continuing to invest large sums of money in information technology. When they tried to identify the payoffs that organizations were obtaining from this investment, however, they were unable to determine whether any had materialized. The question they asked, therefore, was why organizations were continuing to undertake seemingly irrational behavior namely, investing in information technology when little or no payoffs were evident.

Hitt and Brynjolfsson have pointed out how the productivity paradox might be resolved. Whenever we evaluate the impact of investments in information technology, they argue we must address three separate but interrelated questions:

1. Has an investment in information technology increased *productivity* within an organization?

2. Has an investment in information technology increased *profitability* within an organization?

3. Has an investment in information technology created *value for consumers?*

Hitt and Brynjolfsson present evidence that investments in information technology often produce payoffs in terms of productivity and value for consumers but not in terms of organizational profitability. In other words, the benefits of using information technology can be captured in terms of outcomes like employees performing their tasks more efficiently and customers being provided with higher-quality products and services. An

organization might have no increase in profitability, however, as a result of its investments in information technology. The reason is that profitability gains from improved productivity and increased consumer value are quickly competed away. When one organization uses information technology in innovative ways, other organizations often can quickly copy these innovations. As a result, the profitability payoffs obtained from the use of information technology by the innovating organization are short-lived. Nonetheless, if an organization were not to invest in information technology, it might find that its competitive position is eroded. Eventually, it could go out of business. In short, management might have no option but to invest in information technology if they want to stay .in business, even though these investments might produce no improvement in profitability.

Auditors must be careful, therefore, when approaching the evaluation of economic effectiveness for an information system. In particular, they need to address the three questions asked previously. They might have to point out to management why investments in information technology are important,, even though there is no payoff in terms of improved profitability.

In principle, however, there are four steps auditors should seek to undertake when they evaluate the economic effectiveness of an information system:

Step 1: Identify the benefits of the information system

Step 2: Identify the costs of the information system

Step 3: Value the benefits and costs of the information system

Step 4: Determine the net present value of the information system

3. Summary

When evaluating asset safeguarding and data integrity, auditors attempt to determine whether assets could be destroyed, damaged, or used for unauthorized purposes, and how well the completeness, soundness, purity, and veracity of data are maintained. The evaluation process involves auditors making a complex global judgement using piecemeal evidence collected on the strengths and wellness of internal control systems.

The evaluation of system effectiveness involves determining how well a system meets its objectives. The process involves six steps: *(1)* identifying the objectives of the information system, *(2)* selecting the measures to be used, *(3)* identifying data sources, *(4)* obtaining *ex ante* values for the measures, *(5)* obtaining *ex post* values for the measures, and *(6)* assessing the impact of the system by comparing the *ex post* and *ex ante* values of the measures.

Auditors must also recognize that it is sometimes difficult to identify where the benefits and costs from implementing an information system have occurred. For example, information systems often have no payoffs in terms of

improved profitability for an organization. The organization might be more productive, however, and consumers might capture more value from the products and services produced by the organization. The difficulty faced by the organization is that any profitability gains from the information system are quickly competed away. If it does not invest in the information system, however, it might go out of business because it is not competitive.

QUESTIONS

1. List the *six* major steps to be undertaken when evaluating an information system to assess its effectiveness.
2. Give *four* measures of system quality.
3. How might the history of adaptive maintenance for a system affect our assessment of its quality?
4. What is meant by the perceived usefulness of an information system? Give *four* measures of perceived usefulness.
5. How might the source of use of an information system impact an auditor's evaluation of its effectiveness?

REFERENCE BOOKS

1. Weber R; Information Systems Control and Audit (Person Education)
2. Dube: Information systems for Auditing (TMH)
3. Auditing Information Systems, 2nd Edition. Jack J. Champlain (Wiley)