CHAPTER

17

Auditing IT Controls Part III: Systems Development, Program Changes, and Application Auditing

Learning Objectives

After studying this chapter, you should:

- Be familiar with the controls and audit tests relevant to the systems development process.
- Understand the risks and controls associated with program change procedures and the role of the source program library.
- Understand the auditing techniques (CAATTs) used to verify the effective functioning of application controls.
- Understand the auditing techniques used to perform substantive tests in an IT environment.

This chapter is divided into four main sections. The first concludes our treatment of general IT controls as outlined in the COSO control framework. The focus is on Sarbanes-Oxley (SOX) compliance regarding systems development activities and program change procedures. This section examines the risks, controls, audit objectives, and tests of controls that may be performed to satisfy compliance or attest responsibilities. The other three sections of the chapter address tests of application controls and substantive testing in an IT environment. These sections present several computer-aided audit tools and techniques (CAATTs) for testing application controls and conclude with a discussion of embedded audit modules and generalized audit software used for substantive testing.

Systems Development Controls

Chapters 13 and 14 presented the systems development life cycle (SDLC) as a multiphase process by which organizations satisfy their formal information needs. An important point at this juncture is that specific SDLC steps will vary from firm to firm. In reviewing the effectiveness of a particular systems development methodology, the accountant should focus on the controllable activities that are common to all systems development approaches. These are outlined in the following section.

CONTROLLING SYSTEMS DEVELOPMENT ACTIVITIES

This section and the one that follows examine several controllable activities that distinguish an effective systems development process. The six activities discussed deal with the authorization, development, and implementation of new systems. Controls over application changes are presented in the next section.

Systems Authorization Activities

All systems should be properly authorized to ensure their economic justification and feasibility. This requires a formal environment in which users submit requests to systems professionals in written form.

User Specification Activities

Users need to be actively involved in the systems development process. The technical complexity of the system should not stifle user involvement. Regardless of the technology involved, the user should create a detailed written description of his or her needs. The creation of a user specification document often involves the joint efforts of the user and systems professionals. However, this document must remain a statement of user needs. It should describe the user's view of the problem, not that of systems professionals alone.

Technical Design Activities

The technical design activities translate user specifications into a set of detailed technical specifications for a system that meets the user's needs. The scope of these activities includes systems analysis, feasibility analysis, and detailed systems design. The adequacy of these activities is measured by the quality of the documentation that emerges from each phase. Documentation is both a control and evidence of control, and it is critical to the system's long-term success. We discussed specific documentation requirements, including designer, operator, user, and auditor documentation, in Chapter 14.

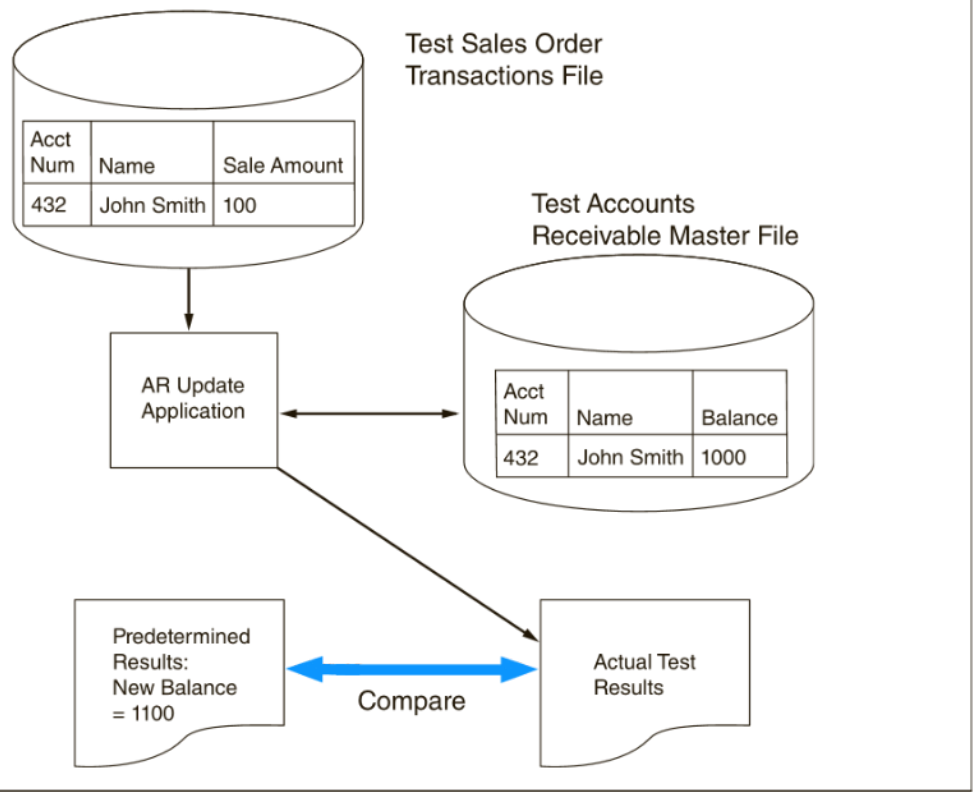
Internal Audit Participation

To meet the governance-related expectations of management under SOX, an organization's internal audit department needs to be independent, objective, and technically qualified. As such, the internal auditor can play an important role in the control of systems development activities. The internal auditor can serve as a liaison between users and the systems professionals to ensure an effective transfer of knowledge. An internal audit group, astute in computer technology and possessing a solid grasp of the business problems to be solved, is invaluable to the organization during all phases of the SDLC. Internal auditors should therefore become formally involved at the inception of the systems development process to oversee the definition of user needs requirements and appropriate controls. Furthermore, this involvement should continue throughout all phases of development and maintenance activities.

Program Testing

All program modules must be thoroughly tested before they are implemented. Figure 17-1 shows a program testing procedure involving the creation of hypothetical master files and transactions files. The results of the tests are then compared against predetermined results to identify programming

FIGURE 17-1 PROGRAM TESTING PROCEDURES



and logic errors. For example, a programmer testing the logic of the accounts receivable update module illustrated in Figure 17-1 might create an accounts receivable master file record for John Smith with a current balance of $1,000 and a sales order transaction record for $100. Before performing the update test, the programmer concludes that a new balance of $1,100 should result. To verify the module's internal logic, the programmer compares the actual results obtained from the test with the predetermined results. This is a very simple example of a program test intended to illustrate the concept. Actual testing would be extensive and involve many transactions that test all aspects of the module's logic.

The task of creating meaningful test data is time-consuming. This should not, however, be considered a single-use activity. As we shall later see, application control testing requires test data. To support future audit needs, test data prepared during systems implementation should be preserved. This will give the auditor a frame of reference for designing and evaluating future audit tests. For example, if a program has undergone no maintenance changes since its implementation, the test results from the audit should be identical to the original test results. Having a basis for comparison, the auditor can thus quickly verify the integrity of the program code. On the other hand, if changes have occurred, the original test data can provide a baseline for assessing the impact of changes. The auditor can thus concentrate tests of application controls on areas where computer logic was changed.

User Test and Acceptance Procedures

Prior to system implementation, the individual modules of the system need to be formally and rigorously tested as a whole. The test team should be composed of user personnel, systems professionals, and internal auditors. The details of the tests performed and their results need to be

formally documented and analyzed. Once the test team is satisfied that the system meets its stated requirements, the system can be transferred to the user.

Many consider the formal testing and acceptance event to be the most important control over the systems development process. This is the last point at which the user can determine the system's acceptability prior to it going into service. Whereas discovering a major flaw at this juncture is costly, discovering it later, during day-to-day operations, may be devastating.

Audit Objectives Relating to Systems Development

The auditor's objectives are to ensure that (1) systems development activities are applied consistently and in accordance with management's policies to all systems development projects, (2) the system as originally implemented was free from material errors and fraud, (3) the system was judged necessary and justified at various checkpoints throughout the SDLC, and (4) system documentation is sufficiently accurate and complete to facilitate audit and maintenance activities.

Tests of Systems Development Controls

The auditor should select a sample of completed projects and review the documentation for evidence of compliance with stated systems development policies. Specific points for review should include determining that:

- User and computer services management properly authorized the project.
- A preliminary feasibility study showed that the project had merit.
- A detailed analysis of user needs was conducted that resulted in alternative conceptual designs.
- A cost-benefit analysis was conducted using reasonably accurate values.
- The detailed design was an appropriate and accurate solution to the user's problem.
- Test results show that the system was thoroughly tested at both the individual module and the total system level before implementation. (To confirm these test results, the auditor may decide to retest selected elements of the application.)
- There is a checklist of specific problems detected during the conversion period, along with evidence that they were corrected in the maintenance phase.
- Systems documentation complies with organizational requirements and standards.

CONTROLLING PROGRAM CHANGE ACTIVITIES

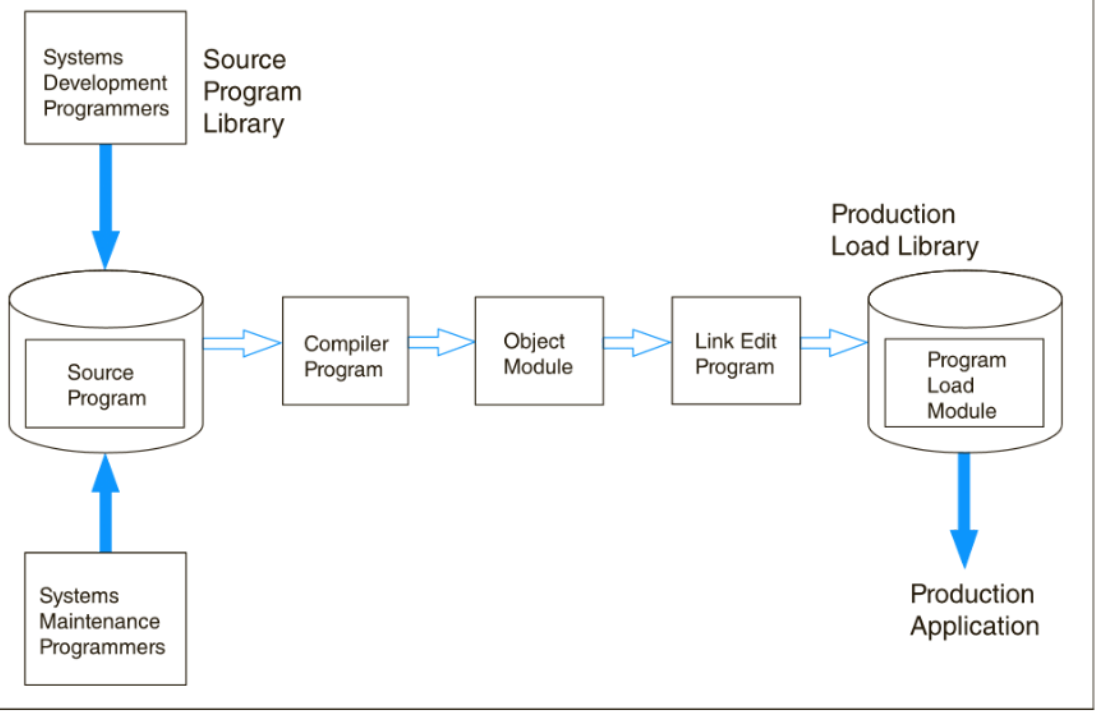
Upon implementation, the information system enters the maintenance phase of the SDLC. This is the longest period in the SDLC, often spanning several years. Most systems do not remain static throughout this period. Rather, they undergo substantial changes that often constitute, in dollars, an amount many times their original implementation cost.

Little is served by designing and implementing controls over systems development activities if control is not continued into the maintenance phase. Unrestricted maintenance access to systems increases the risk that application logic will be corrupted either by accident or intent to defraud. To minimize the risk, all maintenance actions should require, as a minimum, four controls: formal authorizations, technical specifications, testing, and documentation updates. In other words, maintenance activities should be given essentially the same treatment as new development. The extent of the change and its potential impact on the system should govern the degree of control applied. When maintenance causes extensive changes to program logic, additional controls, such as involvement by the internal auditor and additional user test, and acceptance procedures may be necessary.

SOURCE PROGRAM LIBRARY CONTROLS

Even with formal maintenance procedures in place, individuals who gain unauthorized access to programs threaten application integrity. The remainder of this section deals with control techniques and procedures for reducing this risk.

FIGURE 17-2 UNCONTROLLED ACCESS TO THE SOURCE PROGRAM LIBRA



In larger computer systems, application program modules are stored in source code format on a disk repository called the source program library (SPL). Figure 17-2 illustrates the relationship between the SPL and other key components of the operating environment. This material presumes an understanding of the program compilation process. If you are uncertain about the meaning of the terms *source program, compiler,* and *load module,* review the section on language translators on the book's web page, located at www.cengagebrain.com.

Executing a production application requires that the source code be compiled and linked to a load module, which the computer can process. As a practical matter, programs in their compiled state are secure and free from the threat of unauthorized modification. At this point, the source code is not needed for the application to run. In fact, we could destroy it if no future changes to the application were ever to be made. To make a program change, however, requires first changing the logic of the source code on the SPL. This is then recompiled and linked to create a new load module that incorporates the changed code. Clearly, protecting the source code on the SPL is central to protecting the production application.

THE WORST-CASE SITUATION: NO CONTROLS

Figure 17-2 shows the SPL without controls. In this situation, access to application programs is completely unrestricted. Legitimate maintenance programmers and others may access any programs stored in the library, which has no provision for detecting an unauthorized intrusion. Because these programs are open to unauthorized changes, no basis exists for relying on the effectiveness of controls designed into them. Even testing these controls proves only that they work now, but says nothing about how they worked last week or last month. In other words, with no control over access to the SPL, a program's integrity during the period of review cannot be established.

A CONTROLLED SPL ENVIRONMENT

Controlling the SPL requires SPL management system (SPLMS) software. Figure 17-3 illustrates this approach. The black box surrounding the SPL signifies the SPLMS, which controls four critical functions: (1) storing programs on the SPL, (2) retrieving programs for maintenance purposes, (3) deleting obsolete programs from the library, and (4) documenting program changes to provide an audit trail of the changes.

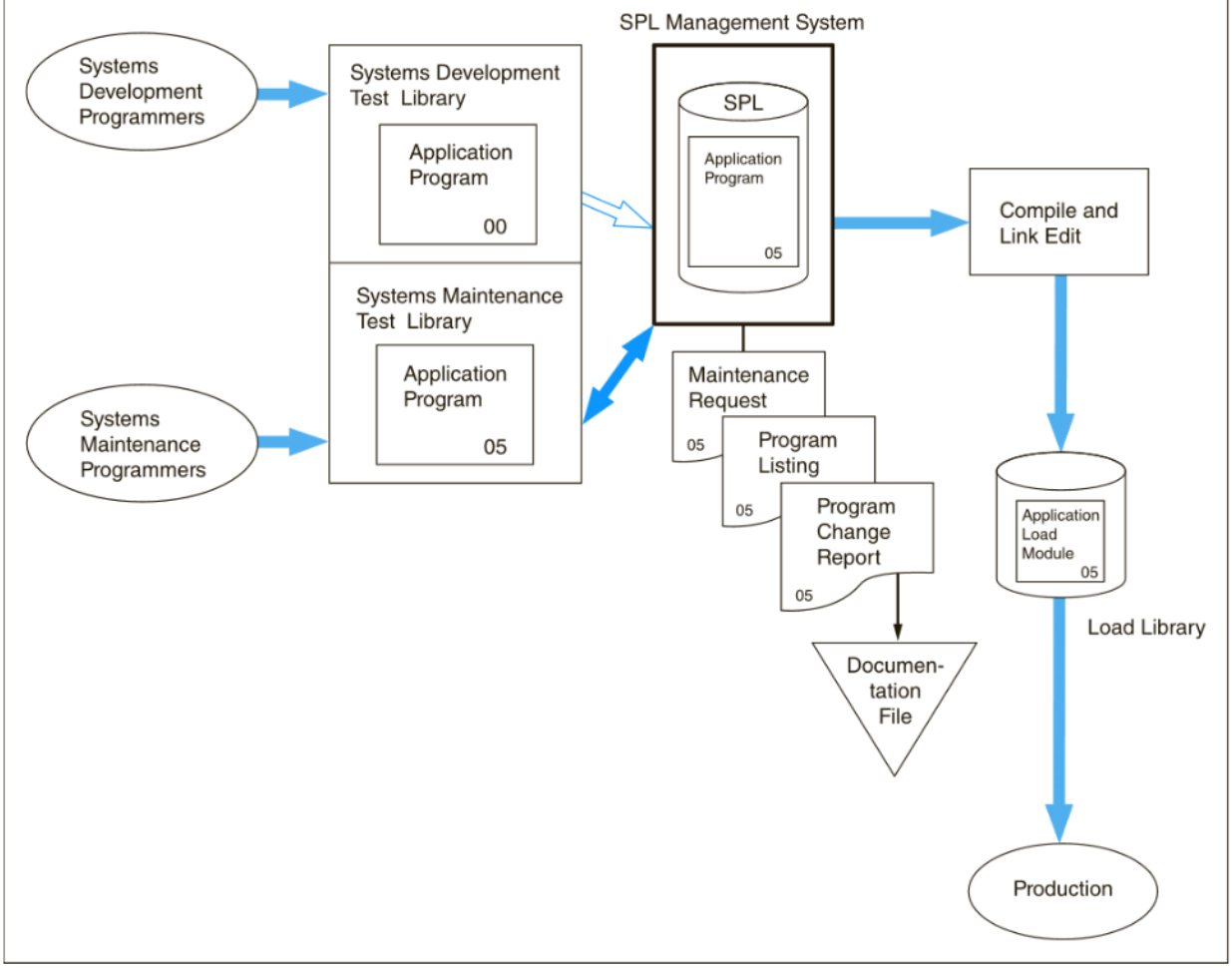
You have probably observed the similarities between the SPLMS and a database management system (DBMS). This is a valid analogy, the difference being that SPL software manages program files and DBMSs manage data files. Computer manufacturers may supply SPLMS software as part of the operating system, or the software may be purchased through third-party vendors.

The mere presence of an SPLMS does not guarantee program integrity. Again, we can draw an analogy with the DBMS. To achieve data integrity, the DBMS must be properly deployed; control does not come automatically—it must be planned. Likewise, an SPL requires specific planning and control techniques to ensure program integrity. The control techniques discussed in the following section address the most vulnerable areas and should be considered minimum SPL controls.

Password Control

Password control over the SPL is similar to password controls in a DBMS. Every financially significant program stored in the SPL can be assigned a separate password. As previously discussed, passwords

FIGURE 17-3 SOURCE PROGRAM LIBRARY UNDER THE CONTROL OF SPL MANAGEMENT SOFTWARE



have drawbacks. When more than one person is authorized to access a program, preserving the secrecy of a shared password is a problem. Because responsibility for the secrecy of a shared password lies with the group rather than with an individual, personal accountability is reduced.

Separation of Test Libraries

Figure 17-3 illustrates an improvement on the shared password approach through the creation of separate password-controlled libraries (or directories) for each programmer. Under this concept, a strict separation is maintained between the production programs that are subject to maintenance in the SPL and those being developed. Production programs are copied into the programmer's library for maintenance and testing purposes only. Direct access to the production SPL is limited to a specific librarian group that approves all requests to modify, delete, and copy programs.

An enhancement to this control feature is the implementation of program-naming conventions. The name assigned to a program clearly distinguishes it as being either a test or a production program. When a program is copied from the production SPL to the programmer's library, it is given a temporary test name. When the program is returned to the SPL, it is renamed with its original production name. This technique greatly reduces the risk of accidentally running an untested version of a program in place of the production program.

Audit Trail and Management Reports

An important feature of SPL management software is the creation of reports that enhance management control and support the audit function. The most useful of these are program modification reports, which describe in detail all program changes (additions and deletions) to each module. These reports should be part of the documentation file of each application to form an audit trail of program changes over the life of the application. During an audit, the reports can be reconciled against program maintenance requests to verify that only approved changes were implemented. For example, if a programmer attempted to use a legitimate maintenance event as an opportunity to commit program fraud, the unauthorized code changes would be documented in the program modification report. These reports can be hard copies or digital and can be governed by password control, thus limiting access to management and auditors.

Program Version Numbers

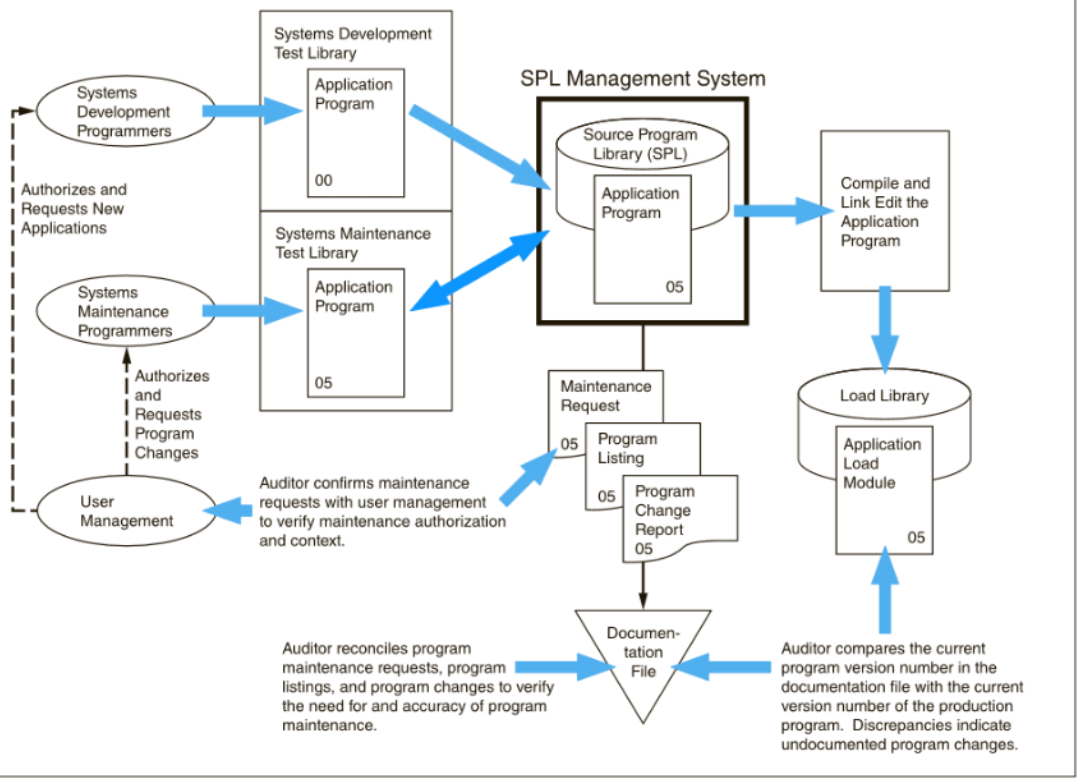
The SPLMS assigns a version number automatically to each program stored on the SPL. When programs are first placed in the libraries (at implementation), they are assigned version number zero. With each modification to the program, the version number is increased by one. For instance, after five authorized maintenance changes, the production program will be Version 05, as illustrated in Figure 17-3. This feature, when combined with audit trail reports, provides a basis for detecting unauthorized changes to the application program. An unauthorized change is signaled by a version number on the production load module that cannot be reconciled to the number of authorized changes. For example, if 10 changes were authorized but the production program is Version 12, then two possible control violations may have happened: (1) authorized changes occurred, which for some reason went undocumented; or (2) unauthorized changes were made, which incremented the version numbers. We will discuss this issue in more detail later.

Controlling Access to Maintenance Commands

Powerful maintenance commands are available for most library systems that can be used to alter or eliminate program passwords, alter the program version number, and temporarily modify a program without generating a record of the modification.

There are a number of legitimate technical reasons why systems designers must sometimes use these commands. If not controlled, however, maintenance commands open the possibility of unauthorized, and perhaps undocumented, program modifications. Hence, access to the maintenance commands themselves should be password controlled, and management or an IT security group should control the authority to use them.

FIGURE 17-4 AUDITING SPL SOFTWARE SYSTEM



Audit Objectives Relating to Systems Maintenance

The auditor's objectives are to determine that (1) maintenance procedures protect applications from unauthorized changes, (2) applications are free from material errors, and (3) program libraries are protected from unauthorized access.

The **tests of controls** necessary to achieve each of these objectives are examined in the following section. The discussion assumes that the organization employs SPL software to control program maintenance. Without such software, achieving the audit objectives may be impossible. The procedures described here are illustrated in Figure 17-4.

Audit Procedures for Identifying Unauthorized Program Changes

To establish that program changes were authorized, the auditor should examine the audit trail of program changes for a sample of applications that have undergone maintenance. The auditor can confirm that authorization procedures were followed by performing the following tests of controls.

RECONCILE PROGRAM VERSION NUMBERS. The permanent file of the application should contain program change authorization documents that correspond to the current version number of the production application. In other words, if the production application is in its tenth version, there should be ten program change authorizations in the permanent file as supporting documentation.[1] Any

1 In most systems, a program in its original (unmodified) state has a version number of 00. Thus, ten changes will give a version number of 10.

discrepancies between version numbers and supporting documents may indicate that unauthorized changes were made.

CONFIRM MAINTENANCE AUTHORIZATION. The program maintenance authorization should indicate the nature of the change requested and the date of the change. The appropriate management from both computer services and the user departments should also sign and approve it. The auditor should confirm the facts contained in the maintenance authorization and verify the authorizing signatures with the managers involved.

Audit Procedures for Identifying Application Errors

The auditor can perform three types of tests of controls—reconcile the source code, review the test results, and retest the program—to determine that programs are free from material errors.

RECONCILE THE SOURCE CODE. Each application's permanent file should contain the current program listing and listings of all changes made to the application. These documents describe in detail the application's maintenance history. In addition, the nature of the program change should be clearly stated on the program change authorization document. The auditor should select a sample of applications and reconcile each program change with the appropriate authorization documents. The modular approach to systems design (creating applications that comprise many small discrete program modules) greatly facilitates this testing technique. The reduced complexity of these modules enhances the auditor's ability to identify irregularities that indicate errors, omissions, and potentially fraudulent programming codes.

REVIEW THE TEST RESULTS. Every program change should be thoroughly tested before being implemented. Program test procedures should be properly documented as to the test objectives, test data, and processing results. The auditor should review this record for each significant program change to establish that testing was sufficiently rigorous to identify any errors.

RETEST THE PROGRAM. The auditor can retest the application to confirm its integrity. We examine several techniques for application testing later in the chapter.

Audit Procedures for Testing Access to Libraries

The existence of a secure program library is central to preventing errors and program fraud. One control method is to assign library access privileges to system librarians only. Their function is to retrieve applications from the program libraries for maintenance and to restore the modified programs to the library. Thus, maintenance programmers test applications in their private libraries but do not have direct access to the program library. The auditor may perform the following tests of controls to assess program library security.

REVIEW PROGRAMMER AUTHORITY TABLES. The auditor can select a sample of programmers and review their access authority. The programmer's authority table will specify the libraries a programmer may access. These authorizations should be matched against the programmer's maintenance authority to ensure that no irregularities exist.

TEST AUTHORITY TABLE. To test the programmer's access privileges, the auditor may violate the authorization rules in an attempt to access unauthorized libraries. The operating system should deny any such attempt.

IT Application Control Testing and Substantive Testing

In addition to general IT controls, SOX requires management and auditors to consider application controls relevant to financial reporting. These controls fall into three broad categories: input controls, processing controls, and output controls. Several previous chapters examined the application

controls pertaining to specific systems, such as sales order processing, purchases, cash disbursements, and payroll systems. Depending upon the system under review and its financial relevance, the auditor will develop specific audit objectives. Once developed, achieving the audit objectives involves performing audit procedures to gather evidence through a combination of tests of application controls and substantive tests of transaction details and account balances. The audit process that relates management assertions, tests of controls, and substantive tests is presented in the appendix to Chapter 15. The remainder of this chapter examines several techniques for performing tests of application IT controls and substantive tests.

DESIGNING TESTS OF APPLICATION CONTROLS

Auditors design tests of IT controls to verify computer logic specific to a particular application. These tests, however, fall into the general categories listed below:

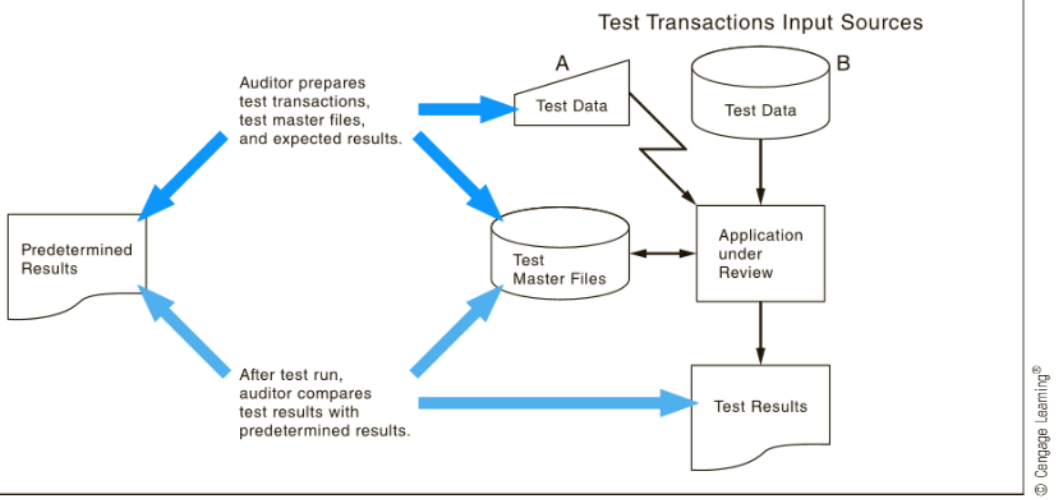
1. **Access tests** verify that individuals, programmed procedures, or messages (such as electronic data interchange [EDI] transmissions) attempting to access a system are authentic and valid. Access tests include verifications of user IDs, passwords, valid vendor codes, and authority tables.
2. **Validity tests** ensure that the system processes only data values that conform to specified tolerances. Examples include range tests, field tests, limit tests, and reasonableness tests. Validity tests also apply to transaction approvals, such as verifying that credit checks and AP three-way-matches are properly performed by the application.
3. **Accuracy tests** ensure that mathematical calculations are accurate and posted to the correct accounts. Examples include recalculations of control totals and reconciliations of transaction postings to subsidiary ledgers.
4. **Completeness tests** identify missing data within a single record and entire records missing from a batch. The types of tests performed are field tests, record sequence tests, and recalculation of hash totals and financial control totals.
5. **Redundancy tests** determine that an application processes each record only once. Redundancy tests include reviewing record counts and recalculation of hash totals and financial control totals.
6. **Audit trail tests** ensure that the application creates an adequate audit trail. Tests include obtaining evidence that the application records all transactions in a transaction log (journal), posts data values to the appropriate accounts, produces complete transaction listings, and generates error files and reports for all exceptions.

Figure 17-5 illustrates the general approach used to test application controls. The steps in this process are listed below:

1) The auditor must first obtain a current version of the application under review or, alternatively, remove the production application from service so that it may be tested directly.
2) Next, the auditor creates test master files containing hypothetical (dummy) records.
3) The auditor then prepares test transactions to be processed through the logic of the application. These transactions should consist of records that are correct, as well as records that contain errors. The error records should be designed such that the internal controls embedded in the application will detect and reject them. If the application processes data in real time, the auditor will submit the transactions via a terminal (option "A" in Figure 17-5). If it is a batch system, then the transaction data will take the form of a transaction file (option "B" in the figure).
4) The next step is for the auditor to calculate the expected results of processing based on the transaction and master file account values assigned to the test data.
5) Finally, the auditor runs the test and reconciles the results obtained with the predetermined results.

Although the above steps seem straightforward in concept, performing them is technically complex and time-consuming. Therefore, to facilitate this activity, several computer-aided auditing

FIGURE 17-5 THE TEST DATA TECHNIQUE



techniques have been developed, which we examine later in the chapter. At this point, we turn our attention to reviewing specific examples of tests of application controls.

Examples of Tests of IT Application Controls

This section presents examples of tests of controls as they apply to specific applications. These tests are representative and not intended to be regarded as a complete set, but the general techniques outlined here may be applied to various transaction-processing tasks.

TESTING CUSTOMER CREDIT APPROVALS. To test credit approval controls in a sales order system, the auditor would create a master file of customer records (AR) with set credit limits. The transactions would consist of sales invoices for amounts that would drive the customer's balances above the credit limit in the master file. If the control is functioning properly, the records will be flagged for management approval or rejected. The sales invoices should also include amounts that do not exceed credit limits. The application should approve and process these transactions against the master file accounts.

TESTING ACCURACY OF POSTINGS TO CUSTOMER ACCOUNTS. This test could be coupled with the previous test with little additional effort and using the same test data. The approved transactions (those that pass the credit limit test) should be accurately posted to the customer accounts in the test master files. The auditor would verify their accuracy by reviewing account balance reports produced by the sales order application and reconcile them with the predetermined results. Discrepancies would indicate a logic error in the math calculation or an error that posts transactions to the wrong accounts.

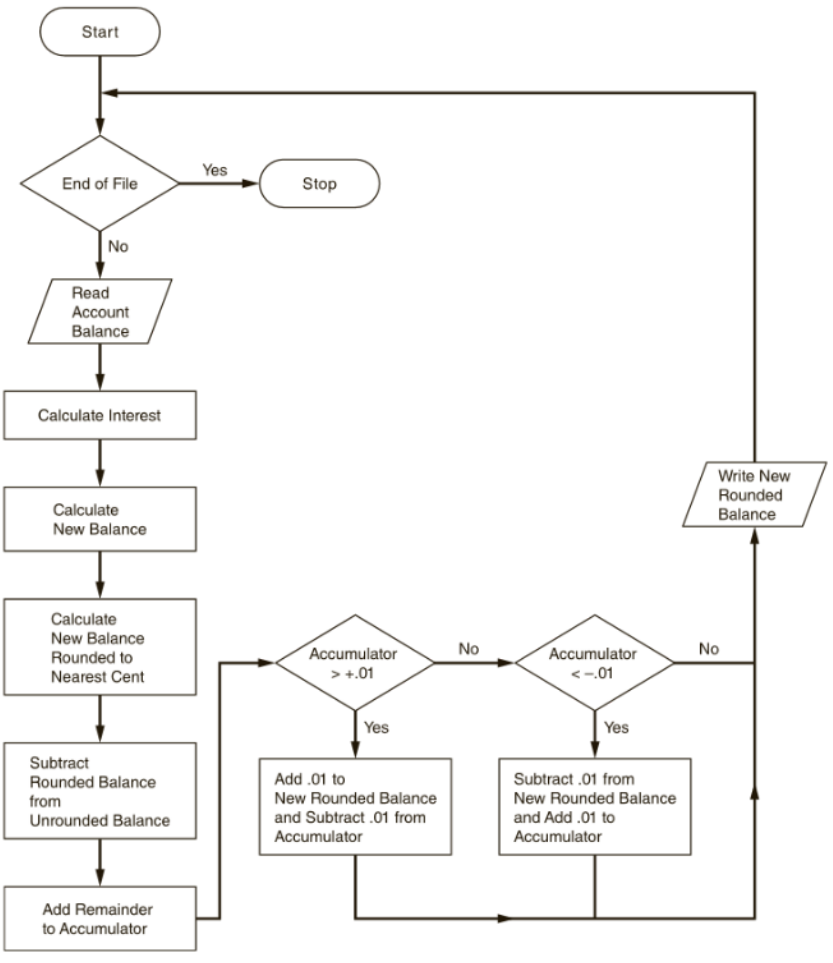
TESTING THE THREE-WAY MATCH. This test involves creating two test master files: a purchase order file and a receiving report file. The transaction in this case is the supplier's invoice. The test data should be designed to contain discrepancies that fall both within and outside of acceptable limits, based on company policy. When the invoice is entered, the AP system should match the three documents (create a digital AP packet) and reconcile the quantities ordered with those received, and the invoice amount with the expected price. The auditor will reconcile both rejected and accepted invoices to determine that the control is functioning in accordance with company policy.

TESTING MULTILEVEL SECURITY AND ACCESS PRIVILEGES IN THE PURCHASES/AP SYSTEM. This test would involve creating several master files: purchase order file; inventory file; receiving report file; and general ledger accounts for cash, inventory control, and AP control. The auditor would then log into the system under different roles and attempt to perform tasks and access data that are not authorized by the various roles. Failure to detect or prevent such attempts indicates a control weakness in the system.

TESTING ROUNDING ERROR ROUTINES IN FINANCIAL SYSTEMS. Financial systems that calculate interest payments on bank accounts or charges on mortgages and other loans employ special rounding error applications. Rounding errors occur when the level of precision used in a calculation is greater than that used for reporting. For example, interest calculations on bank account balances may have a precision of five decimal places, whereas only two decimal places are reported on balances. If the remaining three decimal places are simply truncated, the total interest reported for the total number of accounts will not equal the sum of the individual calculations.

Figure 17-6 shows the logic for handling the rounding error problem. This technique uses an accumulator to keep track of the rounding differences between calculated and reported balances. Note how the sign and the absolute value of the amount in the accumulator determine how

FIGURE 17-6 ROUNDING ERROR ALGORITHM



rounding affects the customer account. To illustrate, the rounding logic is applied to three hypothetical bank account balances (see Table 17-1). The interest calculations are based on an interest rate of 5.25 percent.

Failure to properly account for the rounding difference can result in an imbalance between the total (control) figure and the sum of the detail figures for each account. Rounding errors are also an opportunity for fraud.

SALAMI FRAUD. Rounding programs are particularly susceptible to the so-called **salami fraud**. This fraud affects large numbers of victims, but each in a minimal way. The fraud scheme takes its name from the analogy of slicing a large salami (the total fraud) into many thin pieces. Each victim gets one of these small pieces and is unaware of being defrauded. For example, a programmer, or someone with access to the rounding program shown in Figure 17-6, can modify the rounding

TABLE 17-1 SAMPLE DATA

Record 1	
Beginning accumulator balance	.00861
Beginning account balance	2,741.78
Calculated interest	143.94345
New account balance	2,885.72345
Rounded account balance	2,885.72
Adjusted accumulator balance	.01206 (.00345 + .00861)
Ending account balance	**2,885.73 (round up 1 cent)**
Ending accumulator balance	.00206 (.01206 – .01)
Record 2	
Beginning accumulator balance	.00206
Beginning account balance	1,893.44
Calculated interest	99.4056
New account balance	1,992.8456
Rounded account balance	1,992.85
Adjusted accumulator balance	–.00234 (.00206 – .0044)
Ending account balance	**1,992.85 (no change)**
Ending accumulator balance	–.00234
Record 3	
Beginning accumulator balance	–.00234
Beginning account balance	7,423.34
Calculated interest	389.72535
New account balance	7,813.06535
Rounded account balance	7,813.07
Adjusted accumulator balance	–.00699 (–.00234 – .00465)
Ending account balance	**7,813.07 (no change)**
Ending accumulator balance	–.00699

logic to perpetrate a salami fraud as follows: at the point in the process where the algorithm should increase the current customer's account (that is, the accumulator value is > +.01), the program instead adds one cent to the perpetrator's account. Although the absolute amount of each fraud transaction is small, given the hundreds of thousands of accounts processed, the total amount of the fraud becomes significant over time.

The test of the rounding error control would require special audit software to detect excessive file activity. In the case of the salami fraud, there would be thousands of postings to the computer criminal's personal account, which the audit software should detect. A clever programmer, however, may funnel these entries through several temporary accounts to disguise this activity. These accounts are then posted to a smaller number of intermediate accounts and finally to the programmer's personal account. By using many levels of temporary accounts in this way, the activity to any single account is reduced, and the audit software may not detect it. The auditor can also use audit software to review system audit trail logs (Chapter 16) to detect the existence of unauthorized (dummy) files that contain the intermediate accounts used in such a fraud.

Internal Control Testing Techniques

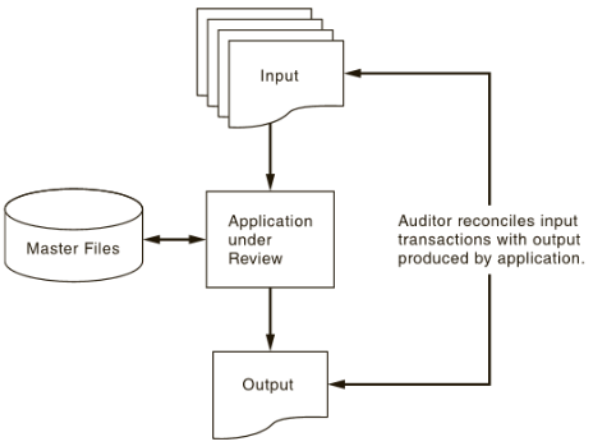
The previous section described the types of application control tests that an auditor may perform. We now examine techniques for performing such tests. We begin by looking at the *black box* approach and then review several *through-the-computer* techniques.

BLACK BOX APPROACH

The **black box approach** (also called *auditing around the computer)* does not require the auditor to create test files or to obtain a detailed knowledge of the application's internal logic. Instead, auditors analyze flowcharts and interview knowledgeable personnel in the client's organization to understand the functional characteristics of the application. With an understanding of what the application is supposed to do, the auditor tests the application by reconciling actual production transactions processed with output results. The output results are analyzed to verify the application's compliance with its functional requirements. Figure 17-7 illustrates the black box approach.

One advantage of this technique is that the application need not be removed from service and tested directly. Black box testing is feasible for applications that are relatively simple, with inputs and outputs that are easily reconciled. More complex applications, however, often draw input data from multiple sources, perform a variety of complex operations, and produce multiple

FIGURE 17-7 AUDITING AROUND THE COMPUTER—THE BLACK BOX APPROACH



outputs. These applications demand more intensive through-the-computer testing to provide the auditor with evidence of application integrity.

THROUGH-THE-COMPUTER APPROACHES

Through-the-computer testing employs **computer-assisted audit tools and techniques (CAATTs)** and requires an in-depth understanding of the internal logic of the application under review. This section describes the key features of five CAATTs: the test data method, base case system evaluation, tracing, integrated test facility, and parallel simulation.

Test Data Method

The **test data method** was introduced previously (see Figure 17-5) to illustrate the general approach to testing application controls. To employ this approach, the auditor requires detailed and current systems documentation: (1) program flowcharts that describe the application's internal logic and allow the auditor to determine which logic branches to test, and (2) record layout diagrams that describe the structure of transaction and master files, which will allow the auditor to create test data.

CREATING TEST DATA. Test data should consist of a complete set of valid and invalid transactions. Incomplete test data may fail to explore critical branches of application logic and error checking routines. Test transactions should be designed to test all possible input errors, logical processes, and irregularities pertinent to the audit objective.

Gaining knowledge of the application's internal logic sufficient to create meaningful test data represents a considerable investment in time. The efficiency of this task is improved, however, through careful planning during systems development. Systems designers should save for future use the test data created to test program modules during the implementation phase of the SDLC. If the application has undergone no changes since its initial implementation, then the current audit test results should equal the original test results obtained at implementation. If the application has been changed, the auditor need only create additional test data that focus on the areas of the program changes.

Results from testing will be in the form of routine output reports, transaction listings, and error reports that are normally produced by the application. In addition, the auditor must examine the updated master files to determine that account balances have been correctly updated. The test results are then compared with the auditor's expected results to determine if the application is functioning properly. This comparison may be performed manually or through the use of special-purpose software.

Using a sales order system as an example, Figure 17-8 illustrates the sort of hypothetical transactions and accounts receivable records that an auditor would prepare. The figure also shows an error report of rejected transactions (the errors are marked with an "X"), and a listing of the updated AR master file. Any deviations between the actual results and those the auditor has predetermined may indicate a logic error or control problem.

Base Case System Evaluation

Base case system evaluation (BCSE) is a variant of the test data approach. BCSE tests are conducted with a set of test transactions containing all possible transaction types. These are processed through repeated iterations during systems development testing until consistent and valid results are obtained. These validated results become the base case. When subsequent changes to the application occur during maintenance, their effects are evaluated by comparing current results with base case results.

Tracing

Another type of the test data technique called **tracing** performs an electronic walk-through of the application's internal logic. The tracing procedure involves three steps:

1. The application under review first undergoes a special compilation process to activate the trace feature.

FIGURE 17-8 EXAMPLE OF TEST DATA AND TEST RESULTS

Test Transaction File

REC NUM	CUST NUM	CUSTOMER NAME	PART NUM	DESCRIPTION	QNTY	UNIT PRICE	TOTAL PRICE
1	231893	Smith, Joe	AX-612	Water Pump	1	20.00	20.00
2	231893	Azar, Atul	J-912	Gear	3	15.00	45.00
3	245851	Jones, Mary	123-LM	Hose	20	20.00	400.00
4	256519	Lang, Tony	Y-771	Spacer	5	2.00	10.00
5	259552	Tuner, Agnes	U-734	Bushing	5	25.00	120.00
6	175995	Hanz, James	EA-74	Seal	1	3.00	3.00
7	267991	Swindle, Joe	EN-12	Rebuilt Engine	1	1,220.00	1,220.00

Original Test AR Master File

CUST NUM	CUSTOMER NAME	CUSTOMER ADDRESS	CREDIT LIMIT	CURRENT BALANCE
231893	Smith, Joe	1520 S. Maple, City	1,000.00	400.00
256519	Lang, Tony	18 Etwine St., City	5,000.00	850.00
267991	Swindle, Joe	1 Shady Side, City	3,000.00	2,900.00

Updated Test AR Master File

CUST NUM	CUSTOMER NAME	CUSTOMER ADDRESS	CREDIT LIMIT	CURRENT BALANCE
231893	Smith, Joe	1520 S. Maple, City	1,000.00	420.00
256519	Lang, Tony	18 Etwine St., City	5,000.00	860.00
267991	Swindle, Joe	1 Shady Side, City	3,000.00	2,900.00

Error Report

REC NUM	CUST NUM	CUSTOMER NAME	PART NUM	DESCRIPTION	QNTY	UNIT PRICE	TOTAL PRICE	EXPLANATION OF ERROR
2	231893	Azar, Atul X	J-912	Gear	3	15.00	45.00	CUSTOMER NAME does not correspond to CUST NUM 231893
3	245851 X	Jones, Mary	123-LM	Hose	20	20.00	400.00	Check digit error in CUST NUM field
5	259552	Tuner, Agnes	U-734	Bushing	5	25.00	120.00 X	Price extension error
6	175995 X	Hanz, James	EA-74	Seal	1	3.00	3.00	Record out of sequence
7	267991	Swindle, Joe	EN-12	Rebuilt Engine	1	1,220.00 X	1,220.00 X	Credit limit error

2. Test data transactions are created.
3. The test data transactions are traced through all processing stages of the program, and a listing is produced of all programmed instructions that were executed during the test.

Figure 17-9 illustrates the tracing process using a portion of the logic for a payroll application. The example shows records from two payroll files—a transaction record showing hours worked and two records from a master file showing pay rates. The trace listing at the bottom of Figure 17-9 identifies the program statements that were executed and the order of execution. Analysis of trace options indicates that Commands 0001 through 0020 were executed. At that point, the application transferred to Command 0060. This occurred because the employee number

FIGURE 17-9 TRACING

Payroll Transaction File

Time Card #	Employee Number	Name	Year	Pay Period	Reg Hrs	OT Hrs
8945	33456	Jones, J.J.	2007	14	40.0	3.0

Payroll Master File

Employee Number	Hourly Rate	YTD Earnings	Dependents	YTD Withhold	YTD FICA
33276	15	12,050	3	3,200	873.62
33456	15	13,100	2	3,600	949.75

Computer Program Logic

```
0001  Read Record from Transaction File
0010  Read Record from Master File
0020  If Employee Number (T) = Employee Number (M)
0030        Wage = (Reg Hrs + [OT Hrs x 1.5] ) x Hourly Rate
0040        Add Wage to YTD Earnings
0050        Go to 0001
0060  Else Go to 0010
```

Trace Listing

0001, 0010, 0020, 0060, 0010, 0020, 0030, 0040, 0050

(the key) of the transaction record did not match the key of the first record in the master file. Then Commands 0010 through 0050 were executed.

Advantages of Test Data Techniques

Test data techniques have three primary advantages. First, they employ through-the-computer testing, thus providing the auditor with explicit evidence concerning application functions. Second, if properly planned, tests can be employed with only minimal disruption to the organization's operations. Third, they require only minimal computer expertise on the part of auditors.

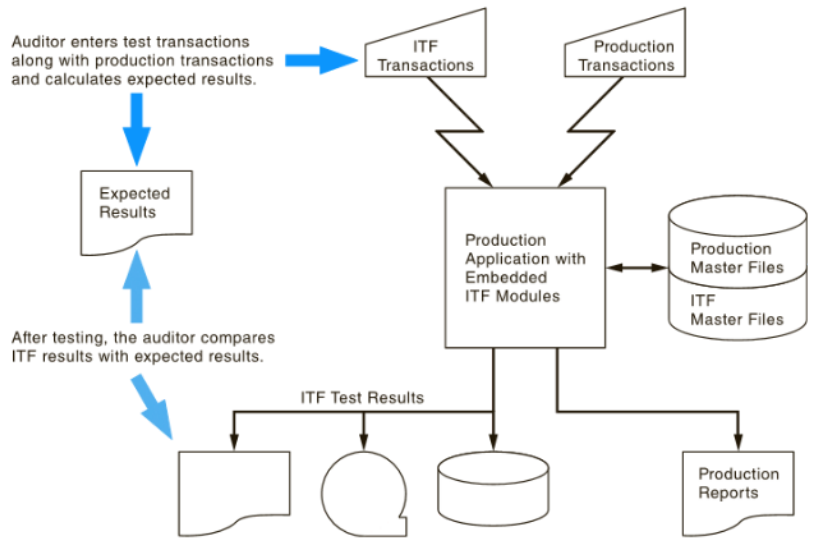
Disadvantages of Test Data Techniques

The primary disadvantage of test data techniques is that auditors rely on the client's IT personnel to obtain a copy of the production application under review. The risk here is that the IT personnel may intentionally or accidentally provide the auditor with the wrong version of the application. Audit evidence collected independently is more reliable than evidence the client supplies. A second disadvantage is that these techniques produce a static picture of application integrity at a single point in time. They are not a convenient means for gathering evidence of ongoing application functionality. A third disadvantage is their high cost of implementation. Gaining an understanding of program logic and creating test data is time-consuming. The following section presents techniques designed to resolve these problems.

THE INTEGRATED TEST FACILITY

The **integrated test facility (ITF)** approach is an automated technique that enables the auditor to test an application's logic and controls during its normal operation. ITF involves designing special audit modules into the application during the systems development process. In addition, corporate production databases are designed to include test master file records. During normal operations, test transactions are merged into the input stream of regular (production)

FIGURE 17-10 THE ITF TECHNIQUE



transactions and are processed against the dummy files in the database. Figure 17-10 illustrates the ITF concept.

ITF audit modules are designed to discriminate between ITF transactions and production data. This may be accomplished in a number of ways. One of the simplest and most commonly used is to assign a unique range of key values exclusively to ITF transactions. For example, in a sales order processing system, account numbers between 2000 and 2100 are reserved for ITF transactions and will not be assigned to actual customer accounts. By segregating ITF transactions from legitimate transactions in this way, ITF test data do not corrupt routine output reports. Test results are produced separately in digital or hard-copy form and distributed directly to the auditor. Just as with traditional test data techniques, the auditor analyzes ITF results against expected results.

Advantages of ITF

The ITF technique has two advantages over test data techniques. First, ITF supports continuous monitoring of controls, as recommended by COSO. Second, ITF-enhanced applications can be economically tested without disrupting the user's operations and without the intervention of computer services personnel. Thus, ITF improves the efficiency of the audit and increases the reliability of the audit evidence gathered.

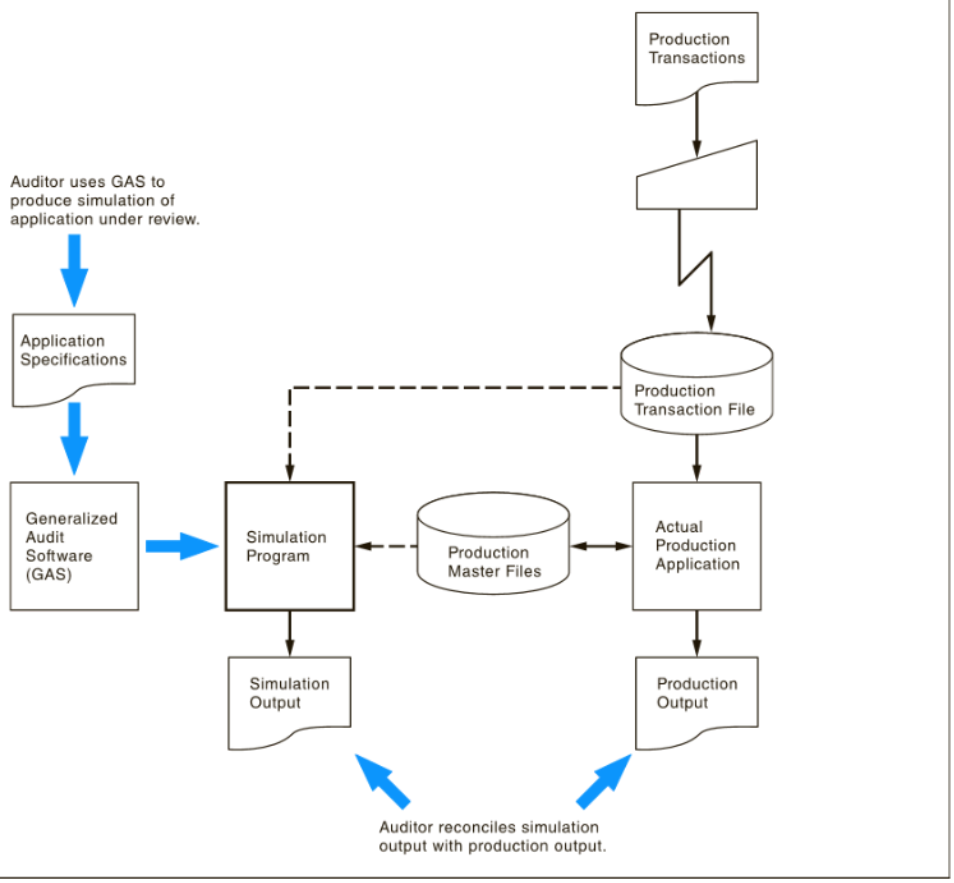
Disadvantages of ITF

The primary disadvantage of ITF is the potential for corrupting corporate databases with test data that may end up in the financial reporting process. Steps need to be taken to ensure that ITF test transactions do not materially affect financial statements by becoming mixed with valid transactions. This problem can be remedied by processing adjusting entries to remove the effects of ITF from general ledger account balances.

PARALLEL SIMULATION

Parallel simulation involves creating a program that simulates key features or processes of the application under review. The simulated application is then used to reprocess the same transactions that the production application previously processed. This technique is illustrated in Figure 17-11. The results

FIGURE 17-11 PARALLEL SIMULATION TECHNIQUE



obtained from the simulation are reconciled with the results of the original production run to determine if application processes and controls are functioning correctly.

Creating a Simulation Program

Simulation packages are commercially available and are sometimes a feature of **generalized audit software (GAS)**.[2] The steps involved in performing parallel simulation testing are outlined in the following section.

1. The auditor must first gain a thorough understanding of the application under review. Constructing an accurate simulation requires complete and current documentation.
2. The auditor must then identify those processes and controls in the application that pertain to the audit objective. These are the processes to be simulated.
3. The auditor creates the simulation using special-purpose commercial software.
4. The auditor runs the simulation program using production transactions and master files to produce a set of results.
5. Finally, the auditor reconciles the test results with the previously created production results.

2 Although GAS can be used for testing internal controls, it is primarily a substantive testing technique. For this reason, this technology is discussed in the following section that deals with substantive testing.

Simulated applications are less complex than the production applications they represent because they contain only processes, calculations, and controls relevant to specific audit objectives. The auditor must therefore carefully interpret the discrepancies between test results and production results. Differences occur for two reasons: (1) the inherent crudeness of the simulation program, and (2) real deficiencies in the application's processes or controls, which the simulation program makes apparent.

Substantive Testing Techniques

Substantive tests are so named because they are used to substantiate dollar amounts in account balances. Substantive tests include, but are not limited to, the following:

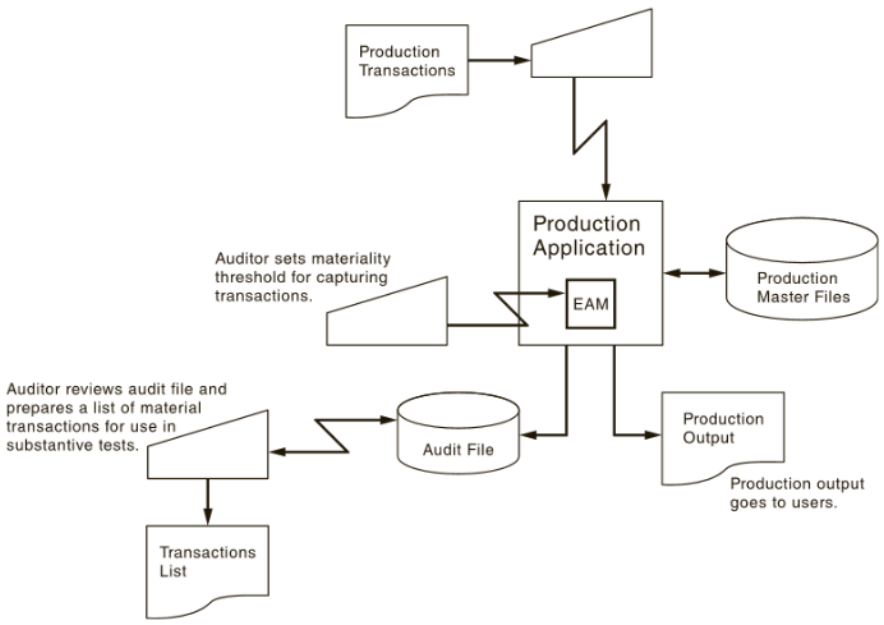
1. Determining the correct value of inventory.
2. Determining the accuracy of prepayments and accruals.
3. Confirming accounts receivable with customers.
4. Searching for unrecorded liabilities.

To perform substantive tests, the auditor must first extract the account data from corporate databases and files. The two CAATTs examined in this section assist auditors in selecting, accessing, and organizing data for substantive tests.

THE EMBEDDED AUDIT MODULE

Embedded audit module (EAM) techniques use one or more programmed modules embedded in a host application to select transactions that meet predetermined conditions. This approach is illustrated in Figure 17-12.

FIGURE 17-12 EMBEDDED AUDIT MODULE TECHNIQUE



As the host application processes the selected transaction, a copy of it is stored on an audit file for subsequent review. The EAM approach allows material transactions to be captured throughout the audit period. The auditor retrieves captured transactions at period-end or at any point in the period, thus significantly reducing the amount of work the auditor must do to identify significant transactions for substantive testing.

To begin data capturing, the auditor specifies to the EAM the parameters and materiality threshold of the transactions set to be captured. For example, assume that the auditor establishes a $50,000 materiality threshold for supplier invoices in an AP system. Invoices equal to or greater than $50,000 will be copied to the audit file. From this set of transactions, the auditor will select a subset to be used for substantive tests. The EAM will ignore transactions that fall below this threshold.

Although primarily a substantive testing technique, EAMs may also be used to monitor application controls on a continuous basis, as recommended in the COSO framework. For example, selected transactions can be reviewed for proper authorization, completeness and accuracy of processing, and correct posting to accounts.

Disadvantages of EAMs

The EAM approach has two significant disadvantages. The first pertains to operational efficiency and the second to EAM integrity.

OPERATIONAL EFFICIENCY. From the user's point of view, EAMs decrease operational performance. The presence of an audit module within the host application may create processing delays, particularly when the level of testing is high. One approach for relieving this burden from the system is to design modules that the auditor may turn on and off. Doing so will, of course, reduce the effectiveness of the EAM as an ongoing audit tool.

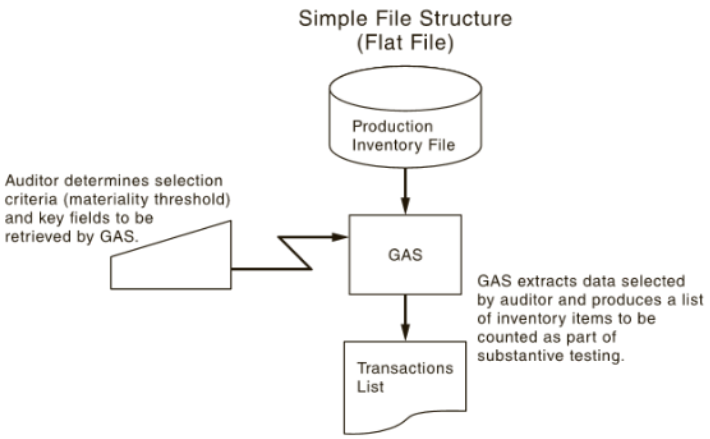
VERIFYING EAM INTEGRITY. The EAM approach may not be a viable audit technique in environments with a high level of program maintenance. When host applications are undergoing frequent changes, the EAMs embedded within the hosts will also require frequent modifications. The integrity concerns raised earlier in the chapter regarding application maintenance apply equally to EAMs. The integrity of the EAM directly affects the quality of the audit process. Auditors must therefore evaluate the EAM integrity. This would be accomplished in the same way as testing the host application controls.

GENERALIZED AUDIT SOFTWARE

Generalized Audit Software (GAS) is a widely used CAATT for IT auditing that allows auditors to access digital data files and perform various operations on the contents. ACL and IDEA are currently the leading GAS products, but others exist with similar features. The following audit tasks can be performed using GAS:

1. Footing and balancing entire files or selected data items.
2. Selecting and reporting detailed data contained on files.
3. Selecting stratified statistical samples from data files.
4. Formatting results of tests into reports.
5. Printing confirmations in either standardized or special wording.
6. Screening data and selectively including or excluding items.
7. Comparing two files and identifying any differences.
8. Recalculating data fields.

FIGURE 17-13 USING GAS TO ACCESS SIMPLE FILE STRUCTURE



The widespread popularity of GAS is due to four factors: (1) GAS languages are easy to use and require little IT background on the part of the auditor, (2) GAS may be used on any type of computer because it is hardware independent, (3) auditors can perform their tests on data independent of client IT professionals, and (4) GAS can be used to audit the data files of many different applications (in contrast with EAMs, which are application-specific).

Using GAS to Access Simple Structures

Accessing flat-file structures (such as a text file) is a simple process, as illustrated in Figure 17-13. In this example, an inventory file is read directly into the GAS, which is configured to extract key information needed for the audit, including the quantity on hand, the dollar value, and the warehouse location of each inventory item. The auditor's task is to perform a physical count of a representative sample of the inventory on hand to verify the existence and value of the inventory. Thus, on the basis of a materiality threshold that the auditor provides, the GAS selects the sample records and prepares a report with the key information.

Using GAS to Access Complex Structures

Accessing complex structures, such as virtual storage access method (VSAM) files and relational database tables, poses more of a problem for the auditor. Most DBMSs, however, have utility features that will reformat complex structures into flat files or spreadsheets, which the GAS can then access. Figure 17-14 shows this technique.

To illustrate the file-flattening process, consider the complex database structure presented in Figure 17-15. The database structure uses pointers to integrate three related files—Customer, Sales Invoice, and Line Item—in a hierarchical model. Extracting audit evidence directly from a structure of this complexity using GAS would be difficult, if not impossible. A simpler flat-file version of this structure is illustrated in Figure 17-16. The resulting single text file represents the three record types as a sequential structure with variable length records that GAS can easily access.

FIGURE 17-14 USING GAS TO ACCESS COMPLEX FILE STRUCTURE

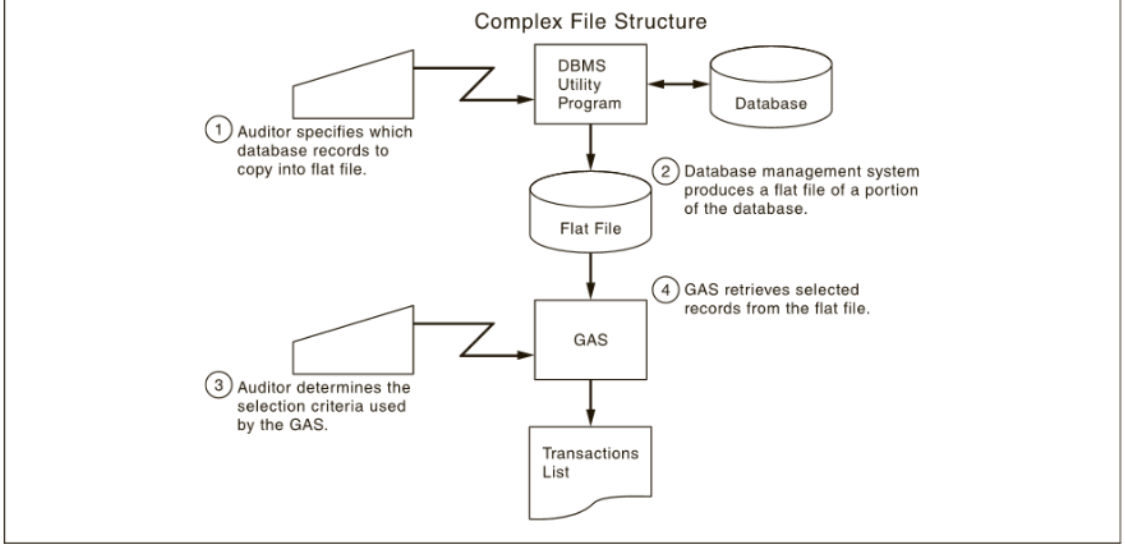
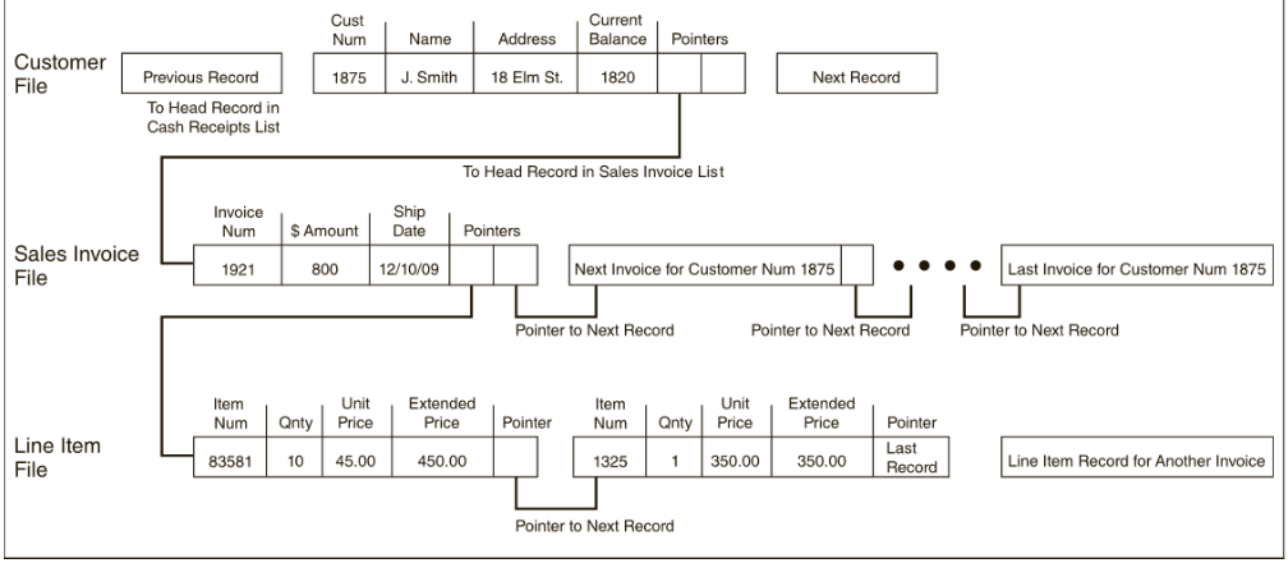


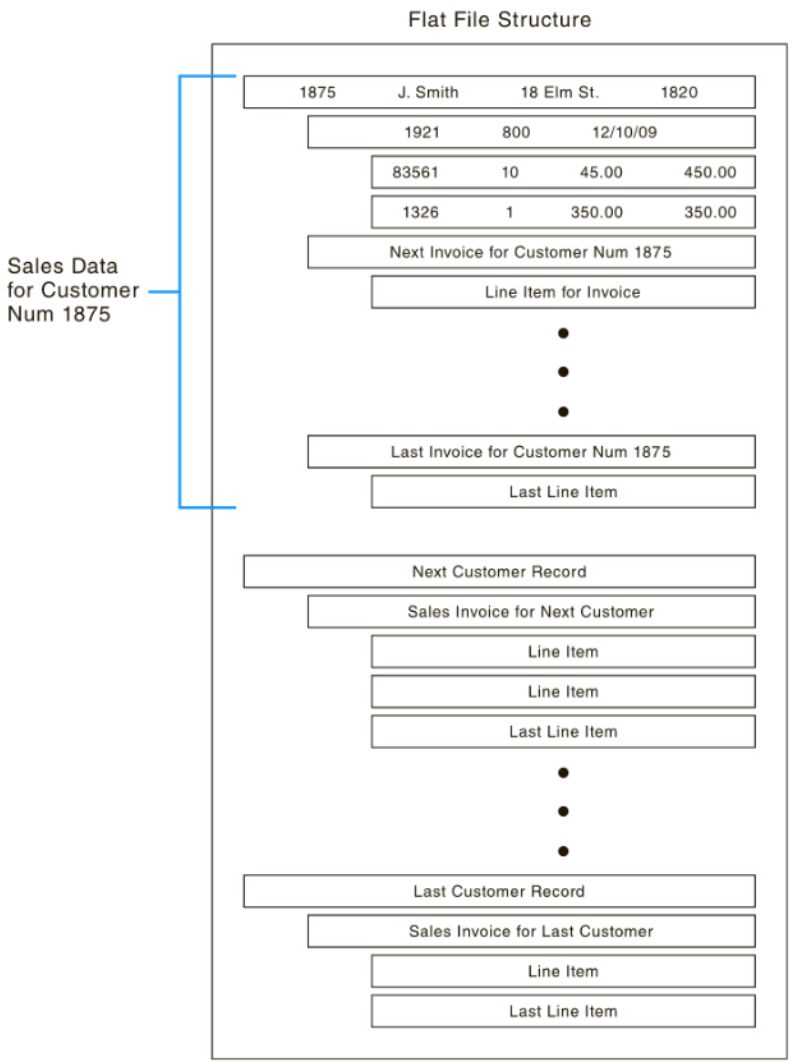
FIGURE 17-15 COMPLEX DATABASE STRUCTURE



Audit Issue Pertaining to the Creation of Flat Files

When auditors rely on client IT personnel to produce flat files from their databases, they run the risk that database integrity will be compromised. For example, if the auditor is confirming accounts receivable, certain fraudulent accounts in the original database may be intentionally omitted from the flat file provided to the auditor. Auditors skilled in relational and object database technology can avoid this problem. Not surprisingly, public accounting firms are aggressively seeking employees with strong computer skills to accompany their accounting training.

FIGURE 17-16 FLAT VERSION OF A COMPLEX FILE STRUCTURE



Summary

This chapter dealt with the business risks, IT controls, and test of controls pertaining to three areas of specific concern to SOX: systems development, program change procedures, and computer applications.

The integrity of financial data is directly dependent on the accuracy of the applications that process them. Likewise, the integrity of those applications depends on the quality of the systems development process that produced them and on the program change procedures through which they were modified. Lack of control over these areas, or inconsistency in their function, can result in unintentional application errors and program fraud.

The systems development and maintenance controls and the test of controls described in this chapter apply both to management's SOX-compliance objectives and the auditor's attest responsibility. To test specific application controls,

auditors (internal and external) use several types of CAATT, including the test data method, the integrated test facility, and parallel simulation. This chapter concluded with a discussion of two popular CAATTs (embedded audit module and generalized audit software) used for substantive testing.

Key Terms

access tests (724)
accuracy tests (724)
audit trail tests (724)
base case system evaluation (BCSE) (729)
black box approach (728)
completeness tests (724)
computer-assisted audit tools and techniques (CAATTs) (729)
embedded audit module (EAM) (734)
generalized audit software (GAS) (733)
integrated test facility (ITF) (731)
parallel simulation (732)
redundancy tests (724)
salami fraud (727)
substantive tests (734)
test data method (729)
tests of controls (722)
tracing (729)
validity tests (724)

Review Questions

1. List the six systems development controls the chapter addresses. List the two systems maintenance controls.
2. Explain how program testing is conducted, and explain the importance of test data.
3. Why are user specification activities important?
4. What is the role of the internal auditor in systems development?
5. What is the purpose of program testing in the SDLC?
6. Give one example of an error that a check digit control detects.
7. Why are program change procedures important to auditors?
8. What is the importance of the SPL?
9. What functions does the SPLMS control?
10. Why should programs undergoing maintenance be renamed?
11. What is a program version number?
12. What tests may be conducted for identifying unauthorized program changes?
13. What tests may be conducted for identifying application errors?
14. What does *auditing around the computer* mean versus *auditing through the computer*? Why is this so important?
15. What are the through-the-computer techniques?
16. What is an embedded audit module?
17. Explain what GAS is and why it is so popular with larger public accounting firms. Discuss the independence issue related to GAS.
18. Name the general categories of IT application control tests that auditors design.
19. What is the purpose of a range check?
20. What is a reasonableness test?

Discussion Questions

1. Discuss how a controlled SPL environment can help to deter unauthorized changes to programs. Can the use of maintenance commands mitigate these controls?
2. What types of output would be considered extremely sensitive in a university setting? Give three examples, and explain why the information would be considered sensitive. Discuss who should and should not have access to each type of information.
3. What are user test and acceptance procedures?