Payoff

Traditionally, firms have adopted stringent systems development controls to ensure that new application systems are efficient and reliable in meeting an organization's and the users' needs. Application change controls have been neglected, however, despite the fact that large firms generally spend between 60% and 80% of their application software dollars on maintenance activities. A new breed of application change control system is emerging that ensures that all changes made to application systems are properly authorized, tested, and approved for implementation. This article examines the design and implementation of these control systems.

Introduction

New application systems are typically developed by in-house programmers in a development environment and then transferred into a production environment to be used to support daily business operations. After being placed in production, however, application systems frequently must be changed to improve the efficiency of the applications, adjust the applications to changing business conditions, or correct defects in the applications. Application changes affect computer programs, screen and file definitions, and Job Control Language instructions, with the bulk of the changes being made to computer programs.

In-house applications systems development and maintenance have different control objectives. Systems development controls provide assurance that application systems are efficient and reliable and meet the organization's and the users' needs. Application change controls ensure that all changes to application components are properly authorized, tested, and approved for implementation. If application change controls are adequate, business users can be confident that the application system being used is the one that was initially developed but with known and approved changes.

In general, large firms spend 60% to 80% of their application software dollars on maintenance activities and the remainder on systems development activities. Each new large systems development project generally is expensive and highly visible; in contrast, each application system change usually is small and low-key. The high visibility and costliness of large systems development projects have compelled firms to adopt stringent systems development controls; however, they have neglected application change controls.

Large systems development controls are grounded in a defined systems development life cycle methodology. The development project is subdivided into phases with tightly controlled milestones, deadlines, coding and testing schedules, and budgets. Moreover, the roles and responsibilities of both programmers and system owners are clearly defined. (A system owner is an individual in a user department who is a liaison between the business users and the programmers and who has stewardship over a particular application system.)

However, substantial systems development controls would yield little value if subsequent modifications to application systems were to undermine these expensive systems development controls. Thus, the integrity of an application system could be in jeopardy without adequate control over application changes. This article describes an emerging new breed of application control systems. Specifically, the article discusses:

- Traditional approaches to application changes.
Traditional Approaches to Application Changes

In the authors' experience as consultants, traditional approaches to application changes are used by many large mainframe computer centers. Firms that do not use contemporary application control procedures store computer programs in common libraries on secondary storage devices. For example, all Common Business Oriented Language source programs (which contain English-like instructions) are stored in a common production source programs library. All COBOL load or machine-executable programs (which contain binary or machine-readable instructions) are stored in a common production load program library.

When creating a new program, the programmer initially codes a source programs and then uses a compiler to generate its load program equivalent. Both programs, after being fully tested, are moved into the common production source and load program libraries, where the load program is executed by the computer to support the organization's daily business operations.

The program change process consists of a series of tasks. When a program change is requested and authorized by the system owner, a programmer copies the source programs from the common production source programs library into the programmer's development source programs library. Development libraries are private libraries that can be created, deleted, read, or written to by a particular programmer and no one else.

After the source programs is in the private development source programs library, the programmer modifies the program, compiles the program to produce a load program, links the program with all essential compiled subroutine, tests the program by executing the load program using test data, and informs the system owner of the successful change. Typically, the system owner approves the change without any additional testing and authorizes the copying of both source and load programs to the common production source and load libraries. The newly generated load program is now used for daily business operations.

Traditionally, organizations have taken two approaches to application changes. One approach provides hundreds of maintenance programmer with update access to the two common production libraries (i.e., source and load) so they can copy programs directly from production to development and back to production at any time. After being given update access, a programmer can read, modify, and delete any program stored in common production libraries.

The main advantage of this approach is efficiency, because programmers can move changed programs to production libraries in a timely fashion without any red tape, burdensome paperwork, or time-consuming bureaucracy. However, this is a dangerous approach to application changes. Because hundreds of maintenance programmers have update access to common production libraries, intentional errors and fatal attacks by disgruntled programmers are difficult to prevent.

The other traditional approach to application changes uses a librarian to perform the task of copying programs from development libraries to production libraries. In this approach, only the librarian has update access to common production libraries. As a result, the risk of attacks is greatly reduced. After a program has been modified successfully and tested by a programmer, forms must be completed authorizing the librarian to move both the source and the load programs from the development libraries to the production libraries.

One disadvantage of this second approach is that moving the changed programs into the production libraries can take several days under this procedure. For emergency changes, the use of a librarian hinders the timely update of production libraries. Therefore, daily
business operations that depend on the successful outcome of these changes may be disrupted.

**Risks Associated with the Traditional Approaches**

In addition to the unique inherent risks in each of the two traditional approaches to application changes, they share several other risks. For example, use of common production libraries causes information privacy problems, because a programmer who has read access to the production source library can read and copy any program in this library. Disgruntled or dishonest programmers can engage in industrial espionage by copying proprietary programs for sale to the competition.

In addition, if only the programmer tests changes, erroneous program instructions can slip into production and the system may fail to meet the needs of the organization and the users. Moreover, having the programming staff perform such incompatible duties as testing changes and moving the changed program to production creates an opportunity for errors and irregularities to occur.

Another risk is that programs are subject to accidental loss if there is no strict version control. Both programmers and the librarian can accidentally copy one program over another and thereby destroy the first program. Moreover, conflicting changes may be made to the same program when two or more programmers concurrently modify the same program without being aware of the other's activities.

In addition, the source programs and its corresponding load program may not be compatible if the load version is copied into the load production library but the source programs is not. Incompatibility between the source and load programs makes future changes to a program much more difficult and expensive.

Because of the risks associated with the traditional approaches to application changes, organizations should consider more effective methods. The following section discusses the design of an improved system for managing application changes.

**Design of an Application Change Control System**

New application change control systems are emerging that mitigate the risks associated with application system changes, thereby protecting the application system's integrity. A common change control system design has also emerged. This section presents an overview of such a design to guide information systems personnel and auditors through the development and implementation of this type of control system.

Application change control systems typically are developed by a joint effort of four groups: system owners from various user departments, application change control administrators, programmers, and auditors. Although control requirements are defined by both system owners and auditors, ease-of-use and efficiency requirements are defined by system owners, change control administrators, and programmers.

A typical application change control system consists of several data bases, libraries, and subsystems (see Exhibit 1). Among the data bases, the control and security data base(shown in the upper right corner of the exhibit) serves as the backbone for defense against unauthorized access to application source program components. It contains the names of the application systems , the name of the owner of each application system , and the names of the programmers who are authorized to make changes to components of each application system.
The historical source data base contains all source components of all the application systems. All authorized programmers have access to this data base; therefore, it is referred to as a common data base. However, read and write privileges are restricted to only those application systems for which certain programmers are authorized.

An activity log data base stores the audit trail. Because all programmers, system owners, and application change control administrators have read access to this data base, it also is called a common data base.

This application change control environment requires three groups of libraries:

- Programmer private libraries.
- System owner acceptance libraries.
- Production libraries.

Management of application system changes is supported by two major subsystems: the user interface subsystem and the administrator interface subsystem. These two subsystems and the other system components are discussed in further detail in the following sections.

**The Control and Security Data Base**

As mentioned previously, the control and security data base contains the application ownership structure of the organization (i.e., the system owners who can authorize changes to be made to their application systems and the programmers who are authorized to make the changes). The top layer of the structure consists of the names of organizational area systems (e.g., accounting, manufacturing, and marketing) and the names of each area's management.

In the middle layer are the individual application subsystems that constitute each organizational area system. The responsibility for keeping each subsystem fully functional is vested in a system owner, who must have an intimate working knowledge of the system. At the bottom layer of the structure are the system owners and authorized maintenance programmer who make changes to application systems on behalf of the system owners.

This application ownership structure governs who can assign system owners and programmers to (or remove them from) an application subsystem. Therefore, update access to this data base must be limited to a few trusted administrators.

**The Administrator Interface Subsystem**

The administrator interface subsystem, which is available only to application change control administrators, is used to perform several controlling tasks. These tasks include:

- Adding application systems to (and removing them from) the application change control system upon management's request.

- Creating and deleting application production and acceptance libraries when application systems are added to or removed from the application change control system.

- Adding maintenance programmers to (and removing them from) an application system upon the system owner's request.

- Deleting application components erroneously added to an application system, upon the system owner's request.
Increasing disk space for application production and acceptance libraries when they become full.

**The Historical Source Data Base**

All source components for application systems are stored in a historical source data base, which consists of two distinct data bases. One data base stores the original source components, and the other stores incremental changes made to the original components. From the two data bases, all prior versions of an application component can be restored quickly by combining the original component with its incremental changes.

The data bases should be fully protected by means of encryption and compression. Furthermore, access to an application component should be restricted to programmers whose identification codes are attached to the application in the control and security data base. Audit trails are logged automatically when these two data bases are updated.

**The Activity Log Data Base**

The activity log data base is used to log all change activities, including:

- Movement of a version of an application component from one environment to another (e.g., production to development, development to acceptance, and acceptance to production).
- Changes made to the historical source and the control and security data bases.
- Actions taken by the system owner of an application (e.g., freeze, approve, or reject).

**Programmer Private Libraries**

Programmer private libraries contain the application components being changed by individual programmers. Only one programmer can create, delete, read, or write to each group of private libraries. In addition, for each application system, there exists a group of system owner acceptance libraries. These libraries contain both source and machine-executable application components that have been tested by programmers and will be tested again by the system owner. Finally, for each application system there is a group of production libraries that contain only machine-executable application components. Thus, there are hundreds of groups of production libraries in this application change control environment, as opposed to just one in the traditional environment.

**System Owner Acceptance Libraries**

System owner acceptance libraries consist of unique groups of libraries for each application system. They contain the application components that have been changed and tested by programmers. These libraries are provided so that testing can be conducted by system owners to reduce the risk of accepting erroneous program instructions inherent in the traditional approaches to program change. As shown in Exhibit 1, the system owner acceptance libraries are sandwiched between the development and production environments.

**Application Production Libraries**

A large organization has hundreds of application systems that are composed of hundreds or even thousands of individual application components. To achieve better
control over application changes, the old common production libraries must be replaced by hundreds of sets of unique application production libraries.

Each set of libraries is used to store the current executable components of a unique application system. Each library within a set is used to store similar components (e.g., all Common Business Oriented Language programs in one library and all Programming Language/1 programs in another library). Access to a set of unique application production libraries should be restricted to authorized programmers, as defined in the application ownership structure.

**The User Interface Subsystem**

Programmers work through the user interface subsystem to invoke the change management subsystem to perform the following tasks:

- Copy application components from the historical source data base to private libraries for making changes.
- Copy changed application components from private libraries to acceptance libraries for acceptance testing conducted by the system owner.
- Move approved, frozen, and changed components from acceptance libraries to the historical data base and production libraries.

System owners can use the user interface subsystem to invoke the change management subsystem to approve and freeze application components being stored in acceptance libraries. Once an application component has been frozen, a programmer can no longer make any more changes to it. Both programmers and system owners can use the ad hoc reporting subsystem to investigate change activities that have taken place.

Because the user interface subsystem allows programmers and system owners to issue instructions to the change management subsystem and the ad hoc reporting subsystem, it should be tailored to the needs and preferences of programmers and system owners. The user interface subsystem must provide for:

- Segregation of duties.
- Source and load program compatibility.
- Elimination of conflicting changes.
- Emergency changes.

The following sections discuss these requirements.

**Segregation of Duties.**

Duties in the user interface subsystem should be segregated between programmers and system owners to prevent errors and irregularities from being both perpetrated and concealed. The user interface subsystem should permit a programmer to instruct the change management subsystem to perform the following tasks:

- Copy a program from the production to the development environment.
- Copy the changed program to the acceptance environment.
· Move the approved, changed program to the production environment.

The user interface subsystem should permit the system owner to instruct the change management subsystem to perform the following tasks:

· Freeze the changed program.
· Approve or reject the changed program.
· Document that acceptance testing has been conducted.

Exhibit 2 shows the proper segregation of incompatible duties. When a requested change to a program is authorized by the system owner, a programmer copies the appropriate source program from the production environment to the programmer's private development library, where the change is made and tested to the programmer's satisfaction. Next, the programmer copies the changed source programs to an acceptance library to allow the system owner to conduct acceptance testing. When the changed source programs is in an acceptance library, the system owner should freeze the changed program to prevent any unexpected modification; the system owner then can test the modified program.

**Segregation of Incompatible Duties Between Programmers and System Owners**

The acceptance testing may have one or two outcomes. If the acceptance testing results are not satisfactory, the system owner can reject the changed source programs and send it back to the programmer. If the acceptance testing results are satisfactory, the system owner can electronically approve the changed program and authorize its movement to the production source programs library.

To achieve total control, the task of moving the changed program should be given to a librarian. However, the use of librarians hinders the timely update of production libraries, as discussed for the second traditional approach. In an application change control environment with many compensatory controls, it is safe and efficient to have the programmer move the changed program.

First, the use of the control and security data restricts the programmers to update only programs for which they are authorized. Second, all prior versions of a program can be quickly restored because the original program and its incremental changes are retained separately in the historical source data base. Third, audit trails are logged automatically when the historical data base is updated. Finally, each application change may affect many components, and it is more difficult to have programmers move the changed program because they have the most knowledge about all the changed components.

**Source and Load Program Compatibility.**

Moving low- and high-level programs to production requires special care to ensure source and load program compatibility. To maintain this compatibility, the user interface subsystem must perform the final compilation of a program when the source programs is copied from the development environment to the acceptance environment. Then, when a program is moved from the acceptance environment to the production environment, the frozen source program and its corresponding load program are moved simultaneously. Thus, source and load program compatibility is ensured, and accidental loss of programs is prevented.
Elimination of Conflicting Changes.

To eliminate conflicting changes in the same program, the user interface subsystem should provide a locking capability to lock a program when it is copied from production to development for modification. The lock should remain in place until the first change has been completed or until a programmer removes it. This control procedure ensures that only one programmer can make a change to a program at a time.

Emergency Changes.

When a defect in an application brings a system down and the system owner is not available to authorize program changes electronically, a programmer must correct the defect quickly to bring the system back into operation. In such emergency situations, the user interface subsystem should permit a programmer to bypass the system owner's electronic authorization to move the changed component to production. The system owner should be notified of the change through electronic mail. Because this type of change is an exception to the normal change control procedure, it must be carefully monitored.

The Change Management Subsystem

The change management subsystem, which is accessed through the user interface subsystem, performs various updating activities to five groups of data bases and libraries: the historical source data base, the activity log data base, the programmer private libraries, the system owner acceptance libraries, and the production libraries.

The Ad Hoc Reporting Subsystem

The ad hoc reporting subsystem, which is accessed through the user interface subsystem, consists mainly of a series of canned parameter-driven programs to permit programmers, system owners, and management to investigate change activities that have taken place. The system must provide the following information, at a minimum:

- The system owner of each application.
- A list of all programs that constitute an application.
- A list of programmers authorized to make changes to an application system.
- Programs being changed at a given time.
- Actions (e.g., freeze, approve, or reject) taken by the system owner of an application.
- Movements of programs from one environment to another (e.g., production to development).
- A list of approved, changed programs that have not been moved to production.
- A list of the users who have received electronic messages regarding emergency changes.
- A log of emergency change attempts made by programmers.
- A list of changed programs that have not been tested by system owners.
Implementing an Application Change Control System

Because implementation of a new application change control system affects all application systems, care must be taken to avoid disruptions of business operations. The main steps involved in implementing a complex application change control system are:

- Development of forms and procedures.
- Purchase or development of software.
- Determination of a library naming convention.
- Conversion.

The following sections discuss these steps in detail.

**Development of Forms and Procedures**

Two forms are needed for an application change control system. One is used by management to add or remove system owners, and the other is used by system owners to add programmers to (or remove them from) the application ownership structure. Next, change control procedures must be developed in detail to define roles and responsibilities of all involved parties, which include programmers, system owners, application change control administrators, and management.

**Purchase or Development of Software**

An application change control system of the type described in this article can be purchased from change control software vendors. Vendors that market change control management products include LEGENT (Westborough MA) and Softool Corp. (Goleta CA). Organizations can buy a system in its entirety or buy portions of it—namely, the change management subsystem and the administrator interface subsystem—and then build the user interface and ad hoc reporting subsystems to reflect the tastes and preferences of the programmers and system owners.

**Determination of a Library Naming Convention**

In the traditional environment, all machine-executable application components of the same type from all application systems are stored in one common production library. In the contemporary environment, they are stored in many unique libraries in addition to the common historical source data base. Because there are hundreds of application systems, hundreds of groups of unique production libraries must be created. Furthermore, acceptance testing requires hundreds of additional groups of acceptance libraries.

To manage the sheer number of acceptance and production libraries, a library naming convention is essential. Each library name must be unique and must consist of a series of the following codes:

- An organizational area code (e.g., MKT for marketing).
- An application system code (e.g., SALES or INVOICE).
- A code indicating whether it is the acceptance or production library.
- The application components type (see Exhibit 3 for a list of types).
A code indicating whether it is the source or load library.

Examples of Application Component Types

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASM</td>
<td>Assembler Program</td>
</tr>
<tr>
<td>COB</td>
<td>COBOL Program</td>
</tr>
<tr>
<td>DBD</td>
<td>IMS Data Base Layout</td>
</tr>
<tr>
<td>FOCEEXEC</td>
<td>FOCUS Program</td>
</tr>
<tr>
<td>FOCMSTR</td>
<td>FOCUS Data Base Layout</td>
</tr>
<tr>
<td>FOCSQL</td>
<td>FOCUS/DB2 Table Layout</td>
</tr>
<tr>
<td>FOR</td>
<td>FORTRAN Program</td>
</tr>
<tr>
<td>JCL</td>
<td>Job Control Language</td>
</tr>
<tr>
<td>PL1</td>
<td>PL/1 Program</td>
</tr>
<tr>
<td>SAS</td>
<td>SAS Program</td>
</tr>
</tbody>
</table>

Exhibit 4 shows two groups of unique libraries (acceptance and production) of an application system. In this exhibit, the organizational area code is MKT, the application system code is SALES, and the acceptance and production indicators are ACPT and PROD. Application component types are FOCEXC, FOCMSTR, and FOCSQL, JCL, SAS, and COBOL. The source and load indicators are SRC and LOAD, respectively.

Libraries of an Application System

<table>
<thead>
<tr>
<th>Acceptance Library</th>
<th>Production Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>MKT.SALES.ACPT.FOCEXEC.SRC</td>
<td>MKT.SALES.PROD.FOCEXEC.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.FOCMSTR.SRC</td>
<td>MKT.SALES.PROD.FOCMSTR.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.FOCSQL.SRC</td>
<td>MKT.SALES.PROD.FOCSQL.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.JCL.SRC</td>
<td>MKT.SALES.PROD.JCL.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.SAS.SRC</td>
<td>MKT.SALES.PROD.SAS.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.COBOL.SRC</td>
<td>MKT.SALES.PROD.COBOL.SRC</td>
</tr>
<tr>
<td>MKT.SALES.ACPT.COBOL.LOAD</td>
<td>MKT.SALES.PROD.COBOL.LOAD</td>
</tr>
</tbody>
</table>

Conversion

Moving components of all application systems from a traditional environment to a contemporary environment is a substantial undertaking for a large computer center. Hundreds of thousands of application system components that are stored together in common libraries must be stored separately in hundreds of unique production libraries. The Appendix at the end of this article presents a checklist of tasks that must be conducted in a massive conversion effort.

Conclusion

Traditional approaches to application changes are inadequate, inefficient, and full of serious exposures to risk. One of the two traditional approaches discussed in this article allows many programmers to read or modify programs stored in common production libraries, thus allowing fatal attacks, mishaps, or espionage to occur. The other approach restricts programmers so much that it can hinder business operations when emergency situations arise. Other risks associated with the two traditional approaches to application changes are:
Inadequate program security.

- Erroneous program instructions.
- Failure of the system to meet the needs of the organization and the users.
- Difficulty of detecting errors and irregularities.
- Accidental loss of programs.
- Source and load program incompatibility.
- Conflicting changes made to the same program at the same time.

The new application change control system described in this article mitigates the risks associated with application changes. Management can build this type of system to be used along with others so that computer-related risks can be reduced to an absolute minimum.

**Author Biographies**

**H. Van Tran**

H. Van Tran, PhD, CPA, is an assistant professor of information systems at the University of Houston-Clear Lake, Houston TX. He has more than 15 years of computer experience in industry and teaches a variety of IS courses, including IS auditing and security. Tran has published several articles in business and professional journals.

**Cynthia D. Heagy**

Cynthia D. Heagy, DBA, CPA, CMA, is an assistant professor of accounting at the University of Houston-Clear Lake. She received a DBA degree from Memphis State University. Heagy has co-written two textbooks—*Accounting Information Systems* and *Bank Accounting and Reporting*—and has published several articles in business and professional journals.