Evaluating Software Risk as Part of a Financial Audit

By Yigal Rechtman

The attention standards setters have given to the computerized environment has varied over the years. Currently, references to information technologies (IT) are sprinkled around the margins of audit standards. Yet in most complex organizations (defined here as a work environment large enough to enable a degree of segregation of duties), the majority of an entity's internal controls are handled by IT. According to a GAO estimate, internal controls are approximately 80% performed in some way by a computerized feature (*Standards for Internal Control in the Federal Government*, November 1999, www.gao.gov/special.pubs/ai00021p.pdf). Reliance on IT controls has become more formally recognized under the Public Company Accounting Oversight Board's (PCAOB) Auditing Standards 2 and 5. AS5 requires the following:

> [27.] As part of evaluating the period-end financial reporting process, the auditor should assess—
>
> ■ Inputs, procedures performed, and outputs of the processes the company uses to produce its annual and quarterly financial statements;
>
> ■ The extent of information technology … involvement in the period-end financial reporting process.

Similarly, Statement on Auditing Standards 109, Understanding the Entity and Its Environment and Assessing the Risks of Material Misstatement, requires auditors to understand the risks in the internal controls environment and specifically assess the effect of IT on these risks.

Generally Accepted Auditing Standards (GAAS) require that "[t]he auditor should also understand how information technology affects relevant control activities" (AU 314.92). Accordingly, "[t]he use of information technology affects the way control activities are implemented." Other portions of GAAS elaborate on this requirement, making sure that auditors are cognizant of the breadth of IT controls as part of internal controls, and of the specific nature of these controls and the special types of risks they present.

Accordingly, auditors should be informed about the risks and requirements with respect to IT-related audit risks. The auditing of automated, pre-programmed, and otherwise technologically implemented controls should be considered in the same way that nonautomated controls are, with more attention given to the risks presented by computerized controls. Auditors should not assume that substantive testing can replace a detection mechanism. This thinking misses a crucial part in understanding how a computerized information system is vulnerable: Electronic records and programs can change instantaneously and without leaving a paper trail. In addition, computer applications can produce a large volume of transactions and balances so large that traditional audit testing would be a few orders of magnitude below the population size. It is thus imperative that auditors properly and fully understand, assess, and respond to the transactions, balances, and disclosures produced by these systems.

The best approach for an auditor is a combination of several types of audit responses: 1) testing of the results of the IT processing; 2) re-performing select transactions; and 3) evaluating the quality of the software used to generate transactions and the amounts that make their way to the financial statements' balances and disclosures. The first two methods are very much like the old-fashioned procedures that enabled auditors to audit around the computer.

The earlier paradigm assumed that computer processing was linear and simple. This view considered IT an "input-process-output" activity, meaning that the validity of the inputs and outputs would ensure the process is valid too. For reasons discussed below, this assumption no longer holds true. Auditors who are interested in assessing and evaluating the quality of IT processing should consider software quality as a primary method of evaluation and assessing control and detection risks within the audit risk assessment process. This article aims to describe and analyze the various methods of assessing IT risks, especially as related to the evaluation of software quality. These methods include benchmarking, regression analysis, reviewing program code, and user acceptance.

Benchmarking

The common thread among early computer applications written before the popularization of the personal computer (PC) age was this: Applications could be described as linear, albeit complex, functions. A linear processing paradigm is one where there is only one single process at any given time. Such a process is simple because it starts with an input that is processed and results in an output; the output is often the input of the next process. Linear processes have one and only one active processing unit at any given time. (By contrast, parallel processing entails multiple processes with varying degrees of interdependence.) The complexities of a business solution were pared down to smaller and simpler programs. Early computer programs were constructed such that the output of one program became an intermediary report as well as the input of the next program. Data structures were also relatively simple; relational databases and data silos were not yet common.

To financial auditors, these intermediate steps, which were often in the form of reconciling reports, gave rise to the concept of auditing around the computer. Because each program processed data and produced some type of report, the auditors could follow the trail of processing from the initial input until the final—and financially material—output. When it came to substantiating the assertions that were produced by these programs, auditors often used benchmarking to track only the changes that had occurred since the last audited period.

For example, suppose that a general ledger application combines monthly journal entries from various areas, such as accounts receivable, accounts payable, and payroll. A legacy software would be written such that each of these modules creates an interface file to the general ledger program (step 1); a utility program scans the interface file and validates its input (step 2); the output file of the utility program becomes an input file for the general ledger program for general journal entries (step 3). In this simplified example, the auditors could simply test the initial, intermediate, and final steps without understanding exactly how the software worked. If there were a change in software (e.g., by adding a department code to the chart of accounts' structure), auditors could benchmark the software by assuming that, except for the one noted change, all functionalities remained the same and, as such, the only significant testing needed would be to ascertain that the correct department was recorded in the monthly journal processing.

As discussed earlier, auditing around the computer is no longer viable, primarily because computer programs are now bundled into software suites and the internal interchanges between modules is not readily tractable. For the same reasons that simple computing models are no longer used, the concept of benchmarking cannot be effectively applied in many cases. Benchmarking can effectively assist the auditors when linear software is used and when the interdependencies between various parts of the software can be very clearly defined. AS5 permits the use of benchmarking as a way to rely on software. Nevertheless, financial and technical auditors should be cognizant of the substantial weaknesses of this audit methodology when a complex input/output environment exists. Not only do the program parameters and logics need to be benchmarked but so do significant underlying operating system libraries, auxiliary system functions, and other programs that operate as part and parcel of the software application.

Auditors should be aware of the prevalence of "libraries" of common functions for a particular software application. The use of libraries almost always negates clear distinctions between parts of the software, even if they can be otherwise logically separated. Libraries are a compilation of commonly used functions that are shared among many modules of the same application. For example, a function that looks up an account in the chart of accounts may be

placed in a library, which will allow all modules of the applications to access it. When the accounts receivable module needs to search for an account number in the chart of accounts, the function in the library is being used. The same function is used when a payroll module requires a lookup in the chart of accounts. Because libraries interconnect several modules with their commonly used functions, the logical separation of these modules is obscured by the common software. When this occurs, the modules become part of one large application that cannot be readily evaluated for changes, making it inappropriate to use benchmarking procedures for assessing risk.

Regression Analysis

Evaluating relationships between observed instances and related results is well documented in statistics. The strength of a relationship between an independent variable and a dependent variable is often measured by R-square. R-square—known as the coefficient of determination—measures how much a set of data varies, and in particular the variation of the dependent variable based on movement of the independent variable. R-square is measured on a scale from –1 to 1 where a –1 represents a perfect inverse relationship and 1 represents a perfect direct relationship; 0 represents no relationship. For example, if the relationship between gross sales and sales commissions paid is 0.85, it means that 85 cents of every dollar of sales commissions paid is related to the magnitude of gross sales.

When it comes to evaluating software, especially a calculation module that is critical and material to the financial statement, the use of regression analysis and R-square can be a helpful tool. Two conditions often need to be satisfied in order for a regression analysis to take place: First, the module must relate materially to a financial statement assertion, and, second, inputs and outputs must be readily available and cost effective. To explain the second condition, note that the effectiveness of regression analysis relies on a high number of observations. A software module that is being tested needs to be "fed" many instances of data and the output needs to be evaluated for all of them. Accordingly, if manual entry is required, the audit procedure is going to be costly. The purpose of the regression testing is to identify anomalies. This is an effective way to identify outputs that do not conform to the specification, especially in a complex software application. When a change is made to the logic of the software, a regression analysis process will re-create real-life scenarios and values and evaluate the system's response. The real-life scenarios must be somewhat random and somewhat consistent, because the goal is to test the revised software for its response, seeking outlying circumstances.

Example. A telephone concern has many billing plans and a software module calculates commissions for the sales force based on any number of variables on a per-contract basis. The software can be modified in such a way that instead of manually entering each contract, it can electronically receive a large quantity of contract information and output the results in an electronically efficient format. The processing is nonlinear: Multiple values and processes funnel into a single result, and the interdependencies make it a complex software application. Several passes of the same data may be required, rendering the software application nonlinear. A regression analysis is then applied to the electronic input and output. The purpose of the regression analysis is to locate possible anomalies in relationships between variables. Suppose that sales commissions are higher for cellular phone contracts with two phones or more. A regression analysis should result in an R-square that approaches 1 for the relationship between the type of contract and number of phones used (independent variables) and the commission paid on these contracts. In a sense, this is a highly complex analytical procedure that does not evaluate the plausibility of the data, but instead the operation of the calculation module. An outlier for this type of a regression analysis would occur where the output does not agree with the specifics designed by the business unit.

The caveat of using a regression analysis as a tool for evaluating software risks is that access to the source code is generally required, as well as a level of cooperation that would allow high-quantity samples of input and output in order to facilitate the procedure. The procedure is best suited for in-house programs and applications, where there is at least one critical module and reliable programming personnel who can facilitate the audit assessment procedure.

To assess the risk related to software, a code review generally includes at least an evaluation of the change management process.

Code Review

Code reviews are a very effective method for establishing the risk of incorrect operation by software that has an effect on a material assertion in the financial statements. Similar to regression analyses, code reviews can be done at several levels and under certain conditions. The key requirement is that the software code is available and technical experts are on hand to evaluate it. Code reviews are sometimes akin to benchmarking because some varieties evaluate not only the changes made but also the process under which such changes were made. To that effect, auditors should be familiar with best practices in the way that software programs and applications are modified. In brief, some of these best practices include documentation of how changes are made, tests performed in noncritical environments, and user acceptance of the final product.

To assess the risk related to software, a code review generally includes at least an evaluation of the change management process. More often, a code review involves some degree of scrutiny of the actual programming language used to develop an application. To accomplish a code review, a human or machine scans the application source programming code to evaluate its susceptibility to known programming errors. One such error can include hardwired values instead of variable, configurable values. For example, a payroll program should not write

the FICA percentage into the source code; if the FICA percentage needs to be changed, finding all the instances where that hardwired value of 7.65% is used will be costly and may render the software unreliable.

If the regular development of the software is undertaken in accordance with software development best practices, auditors can more readily rely on the software's documentation. These often include budgets, initiation forms, technical specifications, and the results of user testing that, in the aggregate, will indicate that good programming practices were undertaken in developing the software. Reviewing the source code with a human eye should be done by an expert programmer. There are also automated solutions that can be configured to search source code for certain types of breaches in programming.

User Acceptance

As part of practices for developing software code, auditors should be aware that initial, internal, and final approval by end users is crucial to the quality of the software. Users are often the subject-matter experts who can most prudently specify and evaluate the uses that the software should have. Even software that is not programmable per se might have changes that are driven by specific values. For example, software may have a fixed logic but contain lookup tables; a mistaken value in the lookup table can cause the software to operate incorrectly, even if the programming logic is correct. One example familiar to most CPAs is payroll software that requires annual updates to the withholding tables and FICA limits.

In situations where the software program code changes, end users' specifications, involvement, and approval are essential to the quality of the resulting software product. When the development process is adequately documented, it provides auditors with a cradle-to-grave audit trail. In most environments, even those that only use configurable software (such as the payroll application example above), user acceptance procedures have a direct effect on the auditor's assessment and the auditor's reliance on the results of the balances and disclosures that the software produces.

Example. A medical billing practice uses industry-specific software to help process claims; the practice also uses the software to generate accounting entries for its own internal use. Assume that there is a change in the formula by which the medical billing practice charges the doctors who are its customers. The software vendor receives an initial request for the change and follows up with a documented flowchart, along with examples of what the new formula would be; the practice reviews the complete kit and formally approves it. The software developer then provides a preliminary program that runs in a test environment. The users are given sufficient time and resources to evaluate the new program in the test environment before formally accepting and approving the change. The software vendor reviews all the test results to prove that the program is operated as intended under the new formula. The software vendor then selects an appropriate date and, in coordination with the users, migrates the new application from the testing environment to the real production environment. Once the new software is installed, users conduct further testing before approving the software in a final formal communication. This type of change management allows all parties involved—from the software developer to the users to regulatory and accounting stakeholders—to be satisfied that all necessary quality control steps have been taken to ensure a successful revision that will serve the user's needs.

Mitigating Risks

Computer applications and information technologies present certain audit risks to even the best-prepared auditor. Many auditors consider IT specialists to be unnecessary because substantive auditing is considered sufficient to compensate for a lack of controls in the computerized environment. This author believes such auditors may be in error. IT systems have become too large and complex—and businesses rely upon them so pervasively—that auditors cannot ignore the audit risks that they represent. Understanding, assessing, and responding to audit risks as they arise from the IT environment is not just a practical matter, it is a necessary part of adhering to auditing standards. ❑

Yigal Rechtman, CPA, CFE, CISM, CITP, *is a director for information technology, technology assurance, and forensic services at Buchbinder Tunick & Co., LLP, New York, N.Y.*