The Sarbanes-Oxley Act of 2002 (SOX) explicitly refers to the validity of the periodic financial reports filed by publicly quoted companies. However the Act is specifically worded, the intention is to bring sanity to the world of financial reporting within companies and between companies and the outside world. Influenced by events such as the Enron and Global Crossing scandals, the U.S. Congress decided it must force company officers to personally declare the truth of the financial reports produced by their companies. Financial accounts such as a corporation’s balance sheet, profit-and-loss statement, and annual report are used both as internal controls and by external parties to guide them in important business decisions, such as “shall I invest in this company?”

SOX attempts to enforce accountability for the “correctness” of these reporting mechanisms by explicitly enforcing accountability at the company officer level. Basically, if company officers sign off on these reports and they turn out to be inaccurate in certain ways, these executives are personally responsible and might go to prison.

There are many hidden traps in the application of SOX relating to consistency within and between the computer systems used to support, provide data to, and produce these financial reports. The initial and obvious targets for scrutiny must be the traditional accounting systems. Clearly, the major financial systems of a company must be carefully built, managed, and audited for the company to conform to SOX. But these systems are fed by other systems that might not initially appear to be targets of SOX.

However, it is quite conceivable they will end up coming under the scope of this act. As an example: an asset-accounting system that has line items relating to the asset value of materials in production is accurate to the extent the production system correctly identifies and assigns value to the materials on the shop floor. Sometimes this is not
easy. Many years ago I worked in the computer department of a steel company in Sheffield, England. One year we performed an actual physical count of steel in the stock yard and in-process and discovered we had a lot less steel on the shop floor than our materials tracking system thought we had. Therefore, because of the deficiencies in the production and materials tracking system, the accounting system was reporting that this company had several million pounds sterling worth of assets more than it actually had. Under SOX, such a discrepancy could result in financial and legal penalties being assessed against the company’s executives.

There are many deficiencies in our ability to develop and maintain systems that may have serious consequences with respect to SOX. The transport of many types of data, asset value, sales records, receivables, outstanding liabilities, and the like may contain errors that compromise the accounting systems and hence the valuation of the company.

The absence of inter-system control mechanisms, consistent systems frameworks, and the lack of auditable sources of data will make it very difficult to assert that the accounts are “accurate”—accurate compared to what? SOX can potentially generate a whole set of control and auditing systems requirements and even separate systems that allow companies to clearly demonstrate their compliance with the Act.

And at some point, we might have to start looking beyond the financials and hard assets and look at software and software development itself.

Software is an Asset Too
A company I work with has, over many years, created a huge system that allows it to integration test the very large communication systems it develops and sells. This test system is clearly a corporate asset. But what is its value? It is a very important resource; otherwise the company would not have built it and would not continue to invest in it. The fact that the test system exists and has the functionality it has allows the company to do things and build things that it could not do and build if the test system did not exist. Its capability even gives it an edge over its competitors who do not have such a system (though most have found they do need to build something similar). But what is it worth?

What Cost, What Price?
There are two traditional approaches to determining value: cost based and price based. The cost-based formula assigns value to an asset based on what it cost to produce, often associated with some time-based depreciation schedule. For most company-internal systems this might be done, though perhaps with some difficulty, since we infrequently keep good records of internal software development costs and there are few defined and accepted formulae available. Even if we do keep these records, costs are rarely integrated into identifiable asset categories in accounting systems. Also, the value of an integration test system is not just what it cost to produce. If the real value of something was simply what it cost us to build it, we probably wouldn’t build it at all, since our return on investment would be zero. Obviously, we’d like to create software artifacts whose value is significantly greater than the cost to build.

The other approach to asset valuation determines value based on how much we could get if we sold the asset. This can also present a problem since, for example, nobody would ever buy the communication integration test system mentioned earlier. This test system only works with the products of this particular company and no other company would find it valuable.

There are accounting ways around this problem. Companies
often use “good will” or a similar category to lump together all the hard-to-account asset value components they own. It is likely that arbitrarily assigning numbers to this category will come under scrutiny by the independent auditors chartered under SOX Title II. Many of the same issues will likely occur when we attempt to assign value to the software asset of a company.

**PROJECTS AS INVESTMENTS**

We encounter a further consideration in “accurate” accounting if we view software development projects as investments. Software projects consume resources and (hopefully) generate returns. They may represent a very significant percentage of the allocated assets of a company. As discussed in my March 2005 column, it is quite appropriate to consider a software project to be equivalent to a type of securities investment. It is interesting to compare the steps and controls companies put on overt financial investments, such as purchasing or issuing bonds, buying other companies, or investing in the stock market, with the rather obvious lack of such controls over the typical software development project. The concept of fiduciary responsibility that governs the management of corporate assets, such as bonds, does not seem to extend to software development projects. The fiscal oversight that is a primary responsibility of company officers stops somewhat short of effectively applying basic principles to systems development.

This is clearly demonstrated in the identification and management of risk on projects. As with other investment vehicles, all projects incorporate some degree of risk. But how much? And how much is enough? Or how much is too little? Companies and company officers rarely have a clear view of this data. Stocks and mutual funds have quoted standard deviations (usually of the unit price variability). One can easily find the \( r^2 \) against an appropriate performance index or other fund, or the beta that indicates the scale of variation against that index. So what is the standard deviation of our projects? Against what benchmark should we measure them? What is the \( r^2 \)? What is the beta? What level of risk are we taking when we commit to developing a system, and is taking that level of risk a responsible act? We need to know, because fiduciary responsibility and appropriate management of a company’s assets requires that we know.

It may be that, in conforming to SOX, companies will find they have to develop answers to these questions and construct systems that assess, track, and report the risk on their software projects. Perhaps we should have been doing this all along.

**POSTSCRIPT: PROJECT INSURANCE**

A company I work with has an insurance corporation as a client. This client company is seriously considering offering indemnity against the failure of software projects. Such a step would have very interesting consequences indeed. To underwrite a software project, the risk and the dollar value of that risk would have to be explicitly quantified. Projects with a poor process, that are under-resourced, are attempting to squeeze a two-year gallon of functionality into a six-month pint pot, are run by overly controlling or capricious management, or any number of a host of development sins that we easily recognize, would find their premiums to be extremely expensive. Well-run projects would get a good deal. Businesses would then have an obvious, unavoidable, and dollar-quoted incentive to change their development practices. Intrinsically high-risk projects would have to be justified by equivalent high return. Companies could set up hedge funds against changing requirements and spread the legitimate risks we incur when we create software systems.

And company officers would be required to build organizations that effectively and demonstrably manage the business of software, and justify their project development decisions or risk being taken away to jail.

**Phillip G. Armour** (armour@corvusintl.com) is a senior consultant at Corvus International Inc., Deer Park, IL, and a research director in the Center for Software Development Innovation at Number Six Software, Arlington, VA.