Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2006 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2006

Compliance-Appropriate Spreadsheet Testing

Raymond Panko
University of Hawaii

Compliance–Appropriate Spreadsheet Testing

Raymond R. Panko
University of Hawaii
Panko@Hawaii.edu

ABSTRACT
Sarbanes–Oxley compliance requirements have forced firms to look at their use of spreadsheets in financial reporting. They are finding that they have many spreadsheets and that testing and other formal development disciplines are rare. The literature on spreadsheet errors has shown that without strong controls, most spreadsheets will have material errors; this means that firms that use uncontrolled spreadsheets cannot plausibly claim to be in compliance with Sarbanes–Oxley. This paper looks at a promising control discipline for bringing spreadsheets into Sarbanes–Oxley compliance, namely logic inspection testing following the traditional Fagan (1976) methodology created for software testing.

Keywords
Spreadsheet, testing, code inspection, logic inspection, Sarbanes–Oxley, compliance, end user computing, human error, error detection.

INTRODUCTION: THE SPREADSHEET CHALLENGE FOR SARBANES–OXLEY COMPLIANCE

In 2002, the U.S. Congress passed the Sarbanes–Oxley Act, which requires strong new corporate controls over financial reporting. Although this was not the first compliance regulation, it drew special attention because it affects all public firms and because it directly affects the chief executive office (CEO) and chief financial officer (CFO) in each firm.

Sarbanes–Oxley requires each firm to assess whether its financial reporting system was effectively controlled during each major reporting period. In addition, an independent auditing firm must concur with or refute this assessment. The requirement for effective control is extremely strict. A firm may not assess its financial reporting system as well controlled if there was even a single material weakness in its financial reporting control system.

The Public Company Accounting Oversight Board (PCAOB) sets requirements for auditing companies. According to the PCAOB (PCAOB 2004), "a material weakness is a significant deficiency or combination of significant deficiencies that result in more than a remote likelihood that a material misstatement of the annual or interim financial statements will not be prevented or detected." Note that Sarbanes–Oxley punishes the possibility of errors, not just actual errors. The threshold for a material error, furthermore, usually is a mere five percent (Vorhies 2005). Under Sarbanes–Oxley, there is almost no margin for control weaknesses.

Failing to effectively control financial reporting can be very costly. Glass, Lewis & Company analyzed 899 cases in which firms reported material weaknesses (Durfee 2005). They discovered that companies experienced an average stock price drop of 4% after an announcement of a material control weakness. In turn, the Dutch research firm ARC Morgan found that in more than 60% of all cases of material weakness reports in 2004, the chief financial officer (CFO) was gone within three months (Durfee 2005)

In response to corporate concerns about Sarbanes–Oxley, research and consulting firms conducted several surveys of how companies actually do financial reporting. What they discovered was spreadsheets—large numbers of spreadsheets. All of these surveys found that at least 80% of all firms were using spreadsheets in financial reporting (www.coda.com, RevenueRecognition.com 2004, Durfee 2004, TMCnet.com 2004, Panko 2006d). Furthermore, when companies use spreadsheets for financial reporting, they often use many. Several firms interviewed informally by the author at a conference on compliance indicated that they used manually operated webs of at least 100 spreadsheets to implement their financial reporting processed. Even firms that use Hyperion or other dedicated financial reporting programs tend to use quite a few spreadsheets and to use them in the riskiest parts of the process, such as end of period adjustments (Debreceny 2005).

Prompted by concerns about liabilities arising from spreadsheets, some companies turned to the research on spreadsheet errors. What they found was extremely troubling. Human error research has shown that human cognition invariably leads to

errors in all cognitive activities (Reason 1990). For cognitive activities of any complexity, such as programming, error rates are inevitably about 2% to 5% (Panko 2006a). (A good example is programming, in which errors (faults) are measured on a per-line of code basis.) Laboratory research has confirmed that similar per-formula error rates exist in spreadsheet development (Panko 2006b), and so have many inspections of real-world operational spreadsheets (Panko 2006c). In large spreadsheets with dozens, hundreds, or even thousands of root (non-copied) formulas, error is inevitable. Inspections of more than 80 operational spreadsheets since 1995 found errors in 94%, despite the fact that most inspections only reported significant errors (Panko 2006c).

How bad are spreadsheet errors? Panko (2006b) interviewed two principals of spreadsheet auditing practices in the United Kingdom, where some spreadsheets must be audited by law. They independently gave data indicating that about 5% of all spreadsheets they audit contain what one of the interviewees called "show stopper" errors. Many more spreadsheets contain smaller but still damaging errors. For example, KPMG (1998) only reported spreadsheets to be incorrect if they contained errors that would make a difference to decision makers. Ninety-one percent of the spreadsheets they examined had errors that large. In turn, a Coopers and Lybrand (1997) study did not report an error unless there was at least a 5% error in a bottom line value. That study found such errors in 91% of all spreadsheets. Recall that five percent it the typical threshold for material errors.

Coopers and Lybrand (1997) did not report an error unless there was at least a 5% error in a bottom line value. That study found such errors in 91% of all spreadsheets. Recall that five percent it the typical threshold for material errors.

Although most first-round Sarbanes-Oxley audits usually focused only on the most glaring control weaknesses, a number of firms did report material deficiencies related to spreadsheets. In 2005, RSM McGladrey examined Securities and Exchange Commission filings and reported that "numerous" companies had already reported control deficiencies due to spreadsheets (Kelly 2005). Future Sarbanes-Oxley audits are likely to be much more rigorous and so are likely to find many more problems.

Given evidence that most spreadsheets probably contain material errors, it is difficult for any company that uses spreadsheets extensively to claim that it does not have to report a material control weakness. Given the widespread use of spreadsheets in financial reporting, it would appear that few firms today can claim that their financial reporting systems are effectively controlled.

Given evidence that most spreadsheets probably contain material errors, it is difficult for any company that uses spreadsheets extensively to claim that it does not have to report a material control weakness. Given the widespread use of spreadsheets in financial reporting, it would appear that few firms today can claim that their financial reporting systems are effectively controlled.

DEALING WITH SPREADSHEETS

There also is a body of research on what companies are doing to control their spreadsheets—or, more correctly, what companies are *not* doing. Surveys have been done on corporate spreadsheet policies and development practices since 1987 (Panko 2006c). All, without exception, have found that few companies have development policies. Furthermore, nearly all developers follow highly risky development practices, including rarely doing comprehensive testing. Spreadsheet development today appears to be where software development was in the 1950s and 1960s.

It seems obvious that better spreadsheet development will be needed to control the use of spreadsheets in Sarbanes–Oxley compliance. Improvements in all phases of the spreadsheet development life cycle will be necessary. These improvements probably can draw from many of the lessons of software engineering, although this is conjecture at this point. Research is badly needed to see how effective various software engineering techniques could be in spreadsheet development. Although a good deal of advice has been written about how to reduce spreadsheet error, few of these prescriptions have been shown to be "safe and effective" on the basis of research. Furthermore, given the paucity of spreadsheet development controls in corporations today, neither experience nor best practices are likely to be good guidelines for improving controls.

This paper focus on a single critical element in spreadsheet development, namely testing. In software development, testing consumes a substantial fraction of all development resources. In a sample of 84 software development projects in 27

organizations, Jones (1998) found that the amount of time spent in testing to reduce errors ranged from 27% to 34%, depending on program complexity. In every case, subjects reported that the project allocated insufficient time for testing (p. 139). In another study, Kimberland (2004) found that Microsoft software development teams spent 40% to 60% of their total working time in testing. We must expect similar resource requirements in spreadsheet inspection

Before testing, programs typically have errors in 2% to 5% of all lines of code (Panko 2006a). After several rounds of testing (unit testing, integration testing, etc.), this error rate falls to about 0.1% to 0.3% (Putnam & Myers 1992). In software development, testing is the primary key to error control, and it seems reasonable to expect this to be the case in spreadsheet development as well

Unfortunately, the most widely cited recommendation on spreadsheet controls for Sarbanes-Oxley compliance (PriceWaterhouseCoopers 2004) lists logic inspection as a control but places this protection tenth on a list of twelve controls. Furthermore, the framework gives almost no information on testing. In fact, its advice on logic inspection—that logic inspection should be done by a single person—flies in the face of what software developers have long known about logic inspection—that a single inspector will not find a high percentage of all errors. This inability of single testers to find a large percentage of errors also has been replicated in spreadsheet inspection experiments (Galletta *et al*. 1993, 1997; Panko, 1999). Fagan (1976) first argued that multi-person code inspection is needed, and subsequent experiments and corporate experience have confirmed this repeatedly

Given the need for controls in spreadsheet development, the likely importance of testing, and the dearth of information in this area, the remainder of this paper will assess the likely usefulness of common testing methods employed in software development. It will focus on one especially promising spreadsheet testing approach that has already begun to be explored somewhat in empirical spreadsheet research. This is logic inspection, which in software development is called code inspection.

TYPES OF TESTING

As noted earlier, testing is not widespread in corporate spreadsheet development. This makes practice-based suggestions about spreadsheet testing highly suspect. In fact, we know that developers are extremely overconfident in the accuracy of their spreadsheets (Panko 2006d) and feel that little testing is necessary. Consequently, they tend to think that superficial testing approaches are acceptable.

Testing in the Systems Development Life Cycle

Every recommended practice about software development makes testing an important activity. However, testing is not a separate stage from code development. Rather, code development and testing are intertwined. In fact, the development of testing plans normally occurs before code development even begins.

During code development, there usually are several phases of testing. For instance, software should be developed in small modules of about 100 to 400 lines of code. Each module should be tested thoroughly. This should be followed by several stages of integration testing, as the program is progressively assembled into larger units.

One type of testing during and after code development is regression testing. Whenever changes are made in a program, regression testing reruns some or all earlier tests to ensure that changes have not created new problems.

Eyeballing

Perhaps the most widely used "testing" approach is to look the results over for reasonableness. This is known as eyeballing. It is a good starting point, but it is wrong to expect much from this approach. Rickets (1990) and Klein, Goodhue, and Davis (1997) have shown that people are not good at seeing even large errors. In compliance, even errors too small to be obvious with eyeballing can cause serious problems. Overall, eyeballing seems is a good beginning step in inspecting a spreadsheet, but it is only that.

Auditing

In the financial accounting literature, auditing means examining a process. Auditors sample only parts of the process to assess if the process appears to be well controlled. While spot-checking is efficient and usually is sufficient for a global assessment of control, it does not catch all or even most errors in the process being audited. In spreadsheet auditing, some analysts propose examining only the risky parts of spreadsheets. While such parts of spreadsheets certainly should be examined especially closely, there are many types of errors in spreadsheet development, and they certainly do not occur only or even

primarily in risky parts of the spreadsheet. For compliance, cost control in error reduction is not a goal. Compliance requires the strongest possible error reduction effort.

Execution Testing

In execution testing, the tester enters input numbers into a module and determines whether the module produces the correct output results. In software development, execution testing is so widespread that it is often seen as synonymous with the word "testing."

Unfortunately, execution testing is very difficult to do without training. Spreadsheet recommendations often speak vaguely about "trying extreme values." However, software testers need weeks of training to learn how to pick test values. They need to know about equivalence classes, code coverage, the need to choose values that should actually break the code, and many other seemingly arcane matters. Without sophistication in the selection of test cases, execution testing has limited value.

In addition, execution testing requires an oracle—a way of determining if a case is correct. In software development, a failure to execute a feature may be the oracle. For spreadsheet development, crashes are rare, so the oracle normally must be some way to tell which of the dozens of tests done on a module is correct. Such oracles rarely exist apart from the spreadsheet calculations themselves.

Another problem is technology. Execution testing in software development can only be done on modules because there are too many input–output permutations for effective execution testing. In software development, usually there are automated ways to input test cases and inspect the results. In spreadsheet development, such technology does not exist. Spreadsheet execution testing for a module needs to input numbers into all input cells, overwriting formulas. If these formulas are not changed back after testing, the spreadsheet will be incorrect in all subsequent uses.

Overall, execution testing seems to be most appropriate for regression testing where past results known to be correct (or at least well-tested). Even then, it will be difficult to apply.

Logic Inspection

Another testing technique used for software development is code inspection, in which a group of programmers independently examines a module of code line by line, looking for problems.

While execution testing is very difficult for untrained personnel, code inspection's simplicity makes it a good potential candidate for end users. In fact, there have been three experiments in spreadsheet logic inspection using inspectors with little training (Galletta et al. 1993, 1997, Panko 1999). Individual inspectors found about 40% to 60% of pre-seeded errors in these inspections. This is about as good as professional programmers do in code inspection.

Logic Inspection Methods

Given the potential for logic inspection for testing, we will look at hypotheses for how logic inspection probably should be done in spreadsheet testing, based on how code inspection has worked in programming and on a few logic inspection experiments on spreadsheets.

Reasonable Expectations

First, it seems important to have reasonable expectations. Many articles about reducing spreadsheet errors talk about eliminating errors. However, testing never eliminates errors in software development or any other aspect of life, such as grammar or spelling (Panko 2006a). Typically, a round of code inspection is likely to eliminate 60% to 80% of all errors in software. This is consistent with what Panko (1999) found in a spreadsheet logic inspection experiment using groups of three subjects.

The Fagan Method

The basic methodology of software code inspection was laid down by Fagan (1976), and it still is the main methodology for code inspection today. In this methodology, there are four phases.

- Initial Meeting. The process begins with an introductory meeting for the team. At this meeting, the code developer describes the requirements of the module and walks the team through the code.
- Individual Inspection. After the meeting, the team members inspect the module individually, looking for errors. There normally is a time limit for this. Often, this time limit is about two hours.

- Ø Concluding Meeting. After the team members have finished their individual inspections, the team holds a concluding meeting at which the team members go over the errors discovered by the individual members and perhaps look for additional errors. They attempt to develop a consensus about which defect reports are real and to rate the severity of defects. They do *not* attempt to fix the errors, however.
- Ø Reporting. One tenet of code inspection is that results should be summarized and reported. Reporting results is critical for learning. Most of what we know about software error rates comes from code inspection reports.

Team Inspection

Why does code inspection use teams rather than individual inspectors? The answer is that research across many cognitive activities has shown that individuals almost never catch all errors (Panko 2006a). This is not due to sloppiness but to limitations in the human cognitive mechanism. Even for finding spelling errors, humans only catch 75% to 90% of all errors (Panko 2006a). For more complex errors, the error detection rate is much lower. Limited error detection ability is simply a fact of human cognition.

Experiments in software code inspection have shown that individual software code inspectors only catch about 40% of all defects (Johnson & Tjahjono 1997, Myers 1978). In industrial settings, even an increase in team size from three to four can make a notable difference (Weller 1993)

In turn, three laboratory experiments have looked at spreadsheet logic inspection. In two studies by Galletta, et al. (1993, 1997) subjects found about half of the seeded errors in a spreadsheet. In the first study, certified public accountants found more errors than people without CPAs, but the gain was small. Experienced spreadsheet developers worked *faster* than subjects without spreadsheet development experience, but they found almost exactly the same percentage of errors.

Panko (1999) adopted the Fagan (1976) methodology to three-person spreadsheet logic inspections. On average, the individuals in his study found 63% of the seeded errors. In the concluding meeting, teams reported 83% of the errors. This was a modest increase, but the gain came primarily from the most-difficult-to-detect errors. This study used a simple spreadsheet based derived from the Galletta et al. (1993) task. In more complex spreadsheets, individual and group detection rates probably will be lower.

Need to Inspect Modules

One tenet of code inspection methodology is to limit the sizes of the modules being tested. Quite simply, if modules are too long, detection yields fall sharply (Barnard & Price 1994). The same is likely to be true in spreadsheet logic inspection, but this needs to be explored in experiments to see if it is indeed true and, if so, how long modules should be.

Speed Hurts

Another tenet of code inspection is the need to inspect slowly. When the speed of inspection increases, yields fall. This has been strongly documented in code inspections of real-world programs (Basili & Perricone 1993, Russell 1991, and Weller 1993).

Human cognition, including error detection, cannot be rushed without disastrous results. As noted earlier, the subjects in the Panko (1999) logic inspection study detected 63% of all errors. They were required to work for 45 minutes, and if they wanted to finish early, they were not allowed to do so. Although the paper did not report it, subjects in another group were allowed to finish when they felt that they had done all they could. These subjects took 32 minutes on average. On average, they only found 52% of the errors.

Management Issues

Logic inspection may be the best choice for spreadsheet testing, but it raises serious management problems. First, who will be in charge of spreadsheet testing (and development) policy? The IT departments barely know that spreadsheets exist and do not have the organizational power to impose discipline on end user developers. In Sarbanes–Oxley compliance, the CFO usually has the power to impose discipline on staff members in financial reporting, but spreadsheet development is done in many parts of the organization.

End user developers, furthermore, only build spreadsheets as part of their job responsibilities. Consequently, they probably will not be responsive to time-consuming professional development. More importantly, they have traditionally had broad discretion in their work. Getting them to develop spreadsheets in a disciplined way in the face of this traditional discretion is

likely to be difficult in itself. Getting them to participate in logic inspection teams will be far more difficult given their traditional freedom and the fact that code inspection usually is painfully unpleasant to do.

CONCLUSION

Given the extensive use of spreadsheets in financial reporting, extensive data on spreadsheet error rates, and the paucity of spreadsheet development and maintenance controls in organizations, it is doubtful that many firms today can plausibly assess their financial reporting systems as free of material weaknesses.

One control that is almost certainly necessary for spreadsheets is testing. In software development, which has the same error rates as spreadsheet development, 30% to 40% of a project's resources must be spent on testing to produce small final error rates. The same is likely to be true in well-controlled spreadsheet development. Consequently, this paper focused on spreadsheet testing. More specifically, it focused on logic inspection, which, the paper argued, is likely to be the most fruitful form of testing in end user development. It discussed, based on software development experience, how the Fagan (1976) methodology for software code inspection is likely to be extensible to spreadsheet testing. It noted that the key elements of software code inspection—such as team inspection rather than individual inspection, the inspection of limited-size modules, and the avoidance of rushed inspection—are likely to be needed in spreadsheet logic inspection. It also noted that empirical research will need to be done to determine how to do logic inspection most effectively for spreadsheets.

At the same time, developing logic inspection methodologies is only one step in controlling spreadsheet development. Other forms of testing also need to be considered, especially execution testing. Other systems development life cycle controls also need to be extended to work with spreadsheet development. Simplistic interpolation of software controls is not likely to be possible because spreadsheets have unique problems, such as the tendency of users to type numbers into formula cells during data entry. This ruins the spreadsheet for later use.

One aspect of spreadsheet control that this paper did not consider at all was spreadsheet *security*. Error reduction can only protect spreadsheets from innocent mistakes. However, spreadsheets also need to be protected from attacks, including fraud to steal money or to cover up problems. This is not just a theoretical concern. Most notably, John Rusnak perpetrated a massive fraud at Allfirst Bank to cover up his currency trading losses (BBC 2002). Rusnak implemented this fraud primarily through spreadsheet manipulation. This fraud went on for several years, ending in 2002 with a cumulative loss of $691 million. Fortunately, several companies have developed spreadsheet security control software in response to the 21 CFR 11 rules imposed on the pharmaceuticals industry by the U. S. Food and Drug Administration to prevent the manipulation of drug test results. These technologies are promising for use in financial spreadsheet development.

REFERENCES

1. Barnard, J., & Price, A. (1994). Managing Code Inspection Information. *IEEE Software, 11(2),* 55-68.
2. Basili, V. R. & Perricone, B. T. (1993). Software Errors and Complexity: An Empirical Investigation. *Software Engineering Metrics, Volume I: Measures and Validation*. Ed. M. Sheppard. Berkshire, England: McGraw-Hill International. 168-183.
3. BBC (2002, October 24). 'Rogue' AIB Trader Pleads Guilty to Fraud. http://news.bbc.co.uk/1/hi/business/2358463.stm.
4. Coopers & Lybrand in London. Description available at http://www.planningobjects.com/jungle1.htm. Contact information is available at that webpage.
5. Debreceny, Roger. (2005). University of Hawaii. Personal communication with the author.
6. Durfee, Don (2004, July/August). SPREADSHEET HELL? *CFO Magazine*, CIO.com, http://www.cfoasia.com/archives/200409-07.htm.
7. Durfee, Don (2005, September 1). The 411 on 404: Reporting Material Weaknesses in Control can Cost Shareholders Millions, *CFO Magazine*, CFO.com. http://www.cfo.com/article.cfm/4315498/c_4334841?f=magazine_alsoinside.
8. Fagan, M. E. (1976). Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal, 15(3),* 182-211.
9. Galletta, D. F.; Abraham, D.; El Louadi, M.; Lekse, W.; Pollailis, Y.A.; & Sampler, J.L. (1993, April-June). An Empirical Study of Spreadsheet Error-Finding Performance. *Journal of Accounting, Management, and Information Technology, 3(2),* 79-95.
10. Galletta, D. F.; Hartzel, K. S.; Johnson, S.; & Joseph, J. L. (1997, Winter). Spreadsheet Presentation and Error Detection: An Experimental Study. *Journal of Management Information Systems,* 13(3).

11. Johnson, P. & Tjahjono, D. (1997, May). Exploring the Effectiveness of Formal Technical Review Factors with CSRS, A Collaborative Software Review System, *Proceedings of the 1977 International Conference on Software Engineering*, Boston, MA.
12. Jones, T. C. (1998). *Estimating Software Costs*. New York: McGraw-Hill.
13. Kelly, Matt (2005, August 23). Spreadsheet Blues: Few Controls Yield Many Weaknesses, *Compliance Week*. http://www.complianceweek.com.
14. Kimberland, K. (2004). Microsoft's Pilot of TSP Yields Dramatic Results. news@sei, No. 2. http://www.sei.cmu.edu/news-at-sei/.
15. Klein, B. D., Goodhue, D. L. & Davis, G. B. (1997). Can Humans Detect Errors in Data? Impact of Base Rates, Incentives, and Goals. *MIS Quarterly, 21(2),* 169-194.
16. KPMG Management Consulting (1998, July 30). Supporting the Decision Maker - A Guide to the Value of Business Modeling. Press Release. http://www.kpmg.co.uk/uk/services/manage/press/970605a.html.
17. McCormick, K. (1983, March). Results of Code Inspection for the AT&T ICIS Project, Paper presented at the Second Annual Symposium on EDP Quality Assurance.
18. Myers, G. J. (1978, September). A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections. *Communications of the ACM, 21(9),* 760-768.
19. Panko, Raymond R. (1999, Fall). Applying Code Inspection to Spreadsheet Testing. *Journal of Management Information Systems, 16(2),* 159-176.
20. Panko, R. R. (2006a). *Human Error Website.* (http://www.cba.hawaii.edu/panko/papers/ss/humanerr.htm). Honolulu, HI: University of Hawai`i.
21. Panko, R. R. (2006b). *Spreadsheet Research (SSR) Website.* (http://www.cba.hawaii.edu/panko/ssr/). Honolulu, HI: University of Hawai`i.
22. Panko, Raymond R. (2006c). Spreadsheets and Sarbanes–Oxley: Regulations, Risks, and Control Frameworks, *Communications of the AIS*. Forthcoming.
23. Panko, Raymond R. (2006d). Two Experiments in Reducing Overconfidence in Spreadsheet Development, *Journal of Organizational and End User Computing*. Forthcoming.
24. PriceWaterhouseCoopers (2004, July). The Use of Spreadsheets: Considerations for Section 404 of the Sarbanes-Oxley. http://www.pwcglobal.com/extweb/service.nsf/8b9d788097dff3c9852565e00073c0ba/cd287e403c0aeb7185256f08007f8caa/$FILE/PwCwpSpreadsheet404Sarbox.pdf.
25. Public Company Accounting Oversight Board (2004, March 17). *Standard No. 2: An Audit of Internal Control Over Financial Reporting Performed in Conjunction with an Audit of Financial Statements*.
26. Putnam, L. H., & Myers, W. (1992). *Measures for Excellence: Reliable Software on Time, on Budget*. Englewood Cliffs, NJ: Yourdon.
27. Reason, J. (1990). *Human Error*. Cambridge, U.K.: Cambridge University Press.
28. RevenueRecognition.com. The Impact of Compliance on Finance Operations. *Financial Executive Benchmarking Survey: Compliance Edition,* 2004. (www.softtrax.com.)
29. Rickets, J. A. (1990, March). Powers-of-Ten Information Biases, *MIS Quarterly, 14(1),* 63-77.
30. Russell, G. W. (1991). Experience with Inspection in Ultralarge-Scale Developments. *IEEE Software, 8(1),* 25-31.
31. TMCnet.com (2004, September 20). European Companies are Taking a Faltering Approach to Sarbanes-Oxley, http://www.tmcnet.com/usubmit/2004/Sep/1074507.htm.
32. Vorhies, J. B (2005, May). The New Importance of Materiality. *Journal of Accountancy* (online). http://www.aicpa.org/pubs/jofa/may2005/vorhies.htm.
33. Weller, M. (1993). Lessons from Three Years of Inspection Data. *IEEE Software, 10(5),* 38-45.