What We Know About Spreadsheet Errors

Article *in* Journal of Organizational and End User Computing · February 2005

DOI: 10.4018/joeuc.1998040102

1 author:

Raymond R. Panko
University of Hawai'i at Mānoa
92 PUBLICATIONS **2,389** CITATIONS

SEE PROFILE

January 2005

A briefer version of this paper
with the same name
has been published in
the *Journal of End User Computing's*
Special issue on Scaling Up End User Development
Volume 10, No 2. Spring 1998, pp. 15-21

What We Know About Spreadsheet Errors

Raymond R. Panko
University of Hawai'i
College of Business Administration
2404 Maile Way
Honolulu, HI 96822
USA

Email: panko@hawaii.edu
Home Page: http://panko.cba.hawaii.edu
Spreadsheet Research (SSR) Website: http://panko.cba.hawaii.edu/ssr/

This Article: http://panko.cba.hawaii.edu/ssr/whatknow.htm

Abstract

Although spreadsheet programs are used for small "scratchpad" applications, they are also used to develop many large applications. In recent years, we have learned a good deal about the errors that people make when they develop spreadsheets. In general, errors seem to occur in a few percent of all cells, meaning that for large spreadsheets, the issue is how many errors there are, not whether an error exists. These error rates, although troubling, are in line with those in programming and other human cognitive domains. In programming, we have learned to follow strict development disciplines to eliminate most errors. Surveys of spreadsheet developers indicate that spreadsheet creation, in contrast, is informal, and few organizations have comprehensive policies for spreadsheet development. Although prescriptive articles have focused on such disciplines as modularization and having assumptions sections, these may be far less important than other innovations, especially cell-by-cell code inspection after the development phase.

Introduction

In a special issue on large system development by end users, it may seem odd to see a paper on spreadsheet development. Spreadsheets are often seen as small and simple "scratch pad" applications, and spreadsheet development is often viewed as a solitary activity. We will see that neither view is correct. Many spreadsheets are large and complex, and development often involves interactions among multiple people. In fact, we would guess that the largest portion of large-scale end user applications today involve spreadsheet development. In addition, the extensive information that has emerged about spreadsheet development may highlight development issues for other technologies. Finally, while spreadsheet development may seem "old," our understanding of spreadsheet development has only matured in the last few years.

The title of this paper is rather presumptuous. It assumes that we have a large enough base of research findings to be able to summarize key results from the relatively young field of spreadsheet errors. Yet we argue that spreadsheet research is indeed sufficiently mature to give useful advice to corporate management and individual developers. There is far more to be done, but we already know enough to speak outside our research field, to working spreadsheet developers and to corporate managers of end user computing. This is good, because the lessons we have learned may require organizations to begin treating spreadsheet development very differently than they have in the past.

A Sense of Urgency

As we will see below, there has long been ample evidence that errors in spreadsheets are pandemic. Spreadsheets, even after careful development, contain errors in one percent or more of all formula cells. In large spreadsheets with thousands of formulas, there will be dozens of undetected errors. Even significant errors may go undetected because formal testing in spreadsheet development is rare and because even serious errors may not be apparent.

Despite this long-standing evidence, most corporations have paid little attention to the prospect of serious spreadsheet errors. However, in 2002, the U.S. Congress passed the Sarbanes-Oxley Act, which requires corporations to have well-controlled financial reporting systems. Even a single instance of a firm having a more-than-remote possibility of a material error or fraud in its financial reporting system will require management and independent auditors to give the firm an adverse opinion on its financial reporting system. This is likely to result in lower stock prices and, in some cases, in fines and prison terms for senior executives. Several other countries have developed or are currently developing similar control requirements for financial reporting systems.

As firms began to prepare for Sarbanes-Oxley and a growing number of other laws requiring the strong control of financial reporting system, they came to realize that many or most financial reporting systems today use spreadsheets and that these spreadsheets are not well-controlled.

- Financial intelligence firm CODA reports that 95% of U.S. firms use spreadsheets for financial reporting according to its experience (www.coda.com).
- In 2004, RevenueRecognition.com (now Softtrax.com), had the International Data Corporation interview 118 business leaders. IDC found that 85% were using spreadsheets in reporting and forecasting and that 85% were using spreadsheets for budgeting and forecasting.
- In 2003, the Hacket Group (www.thehacketgroup.com) surveyed mid-size companies. It found that 47% of companies use stand-alone spreadsheets for planning and budgeting.
- CFO.com interviewed 168 finance executives in 2004. The interviews asked about information technology use within the finance department. Out of 14 technologies discussed, only two were widely used—spreadsheets and basic budgeting and planning system. Every subject said that their department uses spreadsheets.
- In Europe, A.R.C. Morgan interviewed 376 individuals responsible for overseeing SOX compliance in multinationals that do business in the United States. More than 80% of the respondents said that their firms use spreadsheets for both managing the control environment and financial reporting. European Companies

are taking a faltering approach to Sarbanes-Oxley. Sept 20, 2004, TMCnet.com, (http://www.tmcnet.com/usubmit/2004/Sep/1074507.htm)

The question, "What do we do about all the spreadsheets?" is increasingly on the minds of corporate executives.

Although this growing focus on spreadsheet errors within corporations is somewhat gratifying for spreadsheet error researchers, it also requires new research directions in spreadsheet research. In the past, spreadsheet developers have focused most heavily on "innocent errors" caused by human fallibility but not by evil intent. However, to cope with corporate needs raised by Sarbanes-Oxley and other laws, spreadsheet researchers will have to focus on how to develop controls not only for innocent errors but for outright fraud. Existing testing techniques aimed at finding innocent errors are not likely to find fraud-based errors. New forms of paranoid software testing will be needed, and spreadsheet controls will have to work on preventing and identifying fraud. If spreadsheet controls can reduce financial fraud, this would be a major benefit to corporations, because financial fraud is a large current problem. In 2004, fraud through financial statements represented only 7% of all fraud cases studied in an ACFE survey [ACFE, 2004], but the median fraud loss for financial fraud was a million dollars, compared to a median loss of only $100,000 for frauds in general.

Error Rates

Introduction to Error Rates

Since the earliest days of spreadsheeting, there have been concerns about errors in user-developed spreadsheets. Davis (Davis, January 1982) cautioned that "user-developed systems may be hazardous to your organization" in general, because end users seldom implement the disciplines long known to be necessary in the development of systems built with programming languages. With spreadsheet programs, end users became capable of developing analyses containing thousands of cells. Under these conditions, unless the percentage of incorrect cells is almost zero, there will be a very high probability of bottom-line errors.

Over the years, a number of embarrassing spreadsheet development incidents have been reported (Panko, 2005b). In most cases, either the firm that made the error was forced to disclose its mistake or consultants familiar with the error revealed it. Given the normal reluctance of organizations to talk about their mistakes, we should wonder if these incidents were rare cases or whether spreadsheet errors are rather frequent in practice.

More broadly, a number of consultants, based on practical experience, have said that 20% to 40% of all spreadsheets contain errors (Panko, 2005b). In a personal communication with the author, Dent (1995) described an audit in a mining company that found errors in about 30% of the audited spreadsheets. Freeman (Freeman, 1996) cites data from the experience of a consulting firm, Coopers and Lybrand in England, which found that 90% of all spreadsheets with more than 150 rows that it audited contained errors. One Price-Waterhouse consultant audited four large spreadsheets and found 128 errors (Ditlea, 1987).

Today, we have moved beyond such anecdotal evidence, into the realm of systematic field audits and laboratory experiments. Tables 1 through 3 summarize key data from these studies.

Table 1 contains data from 13 field audits involving real-world spreadsheet spreadsheets. Since 1995, when field audits began to use good (although usually not excellent) methodologies, 94% of the 88 spreadsheets audited in 7 studies have contained errors, reflecting improvement in auditing methodologies.

Furthermore, several of these studies only reported major errors.

Table 1: Studies of Spreadsheet Errors

Authors	Year	Number of SSs Audited	Average Size (Cells)	Percent of SSs with Errors	Cell Error Rate	Comment
Davies & Ikin	1987	19		21%		Only serious errors
Cragg & King	1992	20	50 to 10,000 cells	25%		
Butler	1992	273		11%		Only errors large enough to require additional tax payments
Dent	1994	Unknown		30%		Errors caused by users hard-wiring numbers in formula cells. Henceforth, all future computations would be wrong.
Hicks	1995	1	3,856	100%	1.2%	One omission error would have caused an error of more than a billion dollars.
Coopers & Lybrand	1997	23	More than 150 rows	91%		Off by at least 5%
KPMG	1998	22		91%		Only significant errors
Lukasic	1998	2	2,270 & 7,027	100%	2.2%, 2.5%	In Model 2, the investment's value was overstated by 16%. Quite serious.
Butler	2000	7		86%	0.4%**	Only errors large enough to require additional tax payments**
Clermont, Hanin, & Mittermeier	2002	3		100%	1.3%, 6.7%, 0.1%	Computed on the basis of non-empty cells
Interview I*	2003	~36 / yr		100%		Approximately 5% had *extremely* serious errors
Interview	2003	~36 / yr		100%		Approximately 5% had

II*						*extremely* serious errors
Lawrence and Lee	2004	30	2,182 unique formulas	100%	6.9%	30 most financially significant SSs audited by Mercer Finance & Risk Consulting in previous year.
Total since 1995		**88**		**94%**	**5.2%**	

*In 2003, the author spoke independently with experienced spreadsheet auditors in two different companies in the United Kingdom, where certain spreadsheets must be audited by law. Each audited about three dozen spreadsheets per year. Both said that they had never seen a major spreadsheet that was free of errors. Both also indicated that about five percent of the spreadsheets they audited have very serious errors that would have had major ramifications had they not been caught. Audits were done by single auditors, so from the research on spreadsheet and software auditing, it is likely that half or few of the errors had been caught. In addition, virtually all of the spreadsheets had standard formats required for their specific legal purposes, so error rates may have been lower than they would be for purpose-built spreadsheet designs.

**The low cell error rate probably reflects the fact that the methodology did not inspect all formulas in the spreadsheet but focused on higher-risk formulas. However, error has a strong random component, so not checking all formulas is likely to miss many errors.

In turn, Table 2 contains data from experiments involving over a thousand subjects ranging from novices to highly experienced spreadsheet developers. Although there is great diversity in methodology and detailed results, one key pattern stands out clearly: every study that has attempted to measure errors has found them and has found them in abundance.

Table 2: Studies of Spreadsheet Errors

Study	**SSs**	**% with Errors**	**Cell Error Rate (CER)**	**Remarks**
Brown & Gould (1987)	27	63%		9 experienced spreadsheet developers each built 3 SSs. Each developer made at least one error.
Olson & Nilsen (1987-1988)	14		21%	Measured errors as they occurred, even if the subject corrected them. CER only reflects formula cells

Lerch (1988)	21			Measured errors as they occurred, even if the subject corrected them CER only reflects formula cells
All formula cells			9.3%	CER only reflects formula cells
References to cells in the same row			6.4%	CER only reflects formula cells
References to cells in the same column			4.7%	CER only reflects formula cells
References to cells in different rows and columns			14.1%	CER only reflects formula cells Novice programmers each developed several spreadsheets
Hassinen (1988)	92	55%	4.3%	CER only reflects formula cells
Janvrin & Morrison 1st (2000a)	61			Task with 50 links between spreadsheets. CER only reflects some formula cells of high risk--mostly links between spreadsheets Differences were nonsignificant
Working alone, no design training			14%	
Working alone, design training			7%	
Working in pairs, no design training			7%	
Working in pairs, design training			7%	
Janvrin & Morrison 2nd (2000b)				Task with 66 links CER only reflects some formula cells of high risk--mostly links between

				spreadsheets Differences were nonsignificant
No training in design	30		16.8%	
Training in design	58		8.4%	
Kreie, et al. (post test) (2000)	73	42%	2.5%	
Teo & Tan (1997)	168	42%	2.1%	Wall task.
Panko & Halverson (1997)				Galumpke task. Undergraduates Based on all text and number cells
Working alone	42	79%	5.6%	
Dyads (Groups of 2)	46	78%	3.8%	
Tetrads (Groups of 4)	44	64%	1.9%	
Panko & Halverson (2001)	35	86%	4.6%	Undergraduates
Panko & Halverson (2001)				
Panko & Sprague (1998)	26	35%	2.1%	Wall Task. MBA students with little or no SS development experience
Panko & Sprague (1998)	17	24%	1.1%	Wall Task. MBA students with 250 hours or more SS development experience

---

Finally, Table 3 looks at code inspection experiments. In these experiments, subjects examined spreadsheets seeded with errors. The research shows that people are not only prone to make errors. They are only moderately good at correcting these errors.

Table 3: Code Inspection Experiments

Study	Subjects	Sample Size	% of Errors Detected	Remarks
Galletta et al. (1993)	MBA students & CPAs Taking a Continuing Education Course			Budgeting task containing seeded errors
Total sample		60	56%	
CPA novices	<100 hours of work experience with SSs	15	57%	
CPA experts	<100 hours of work experience with SSs	15	66%	
MBA students, novices	>250 hours of work experience with SSs	15	52%	
MBA students, experienced	>250 hours of work experience with SSs	15	48%	
Domain (logic) errors corrected			46%	
Device (mechanical) errors corrected			65%	
Galletta et al. (1997)	MBA students		51%	Same task used 1993 study
Overall		113	51%	
On-Screen		45	45%	
On Paper		68	55%	
Panko (1999)				Modified version of Galletta wall task.
Undergrads working alone		60	63%	
Undergrads working in groups of three		60	83%	
Panko & Sprague (1998)	Undergrads	23	16%	Students who made errors in the Wall task who then went on to inspect their own spreadsheets.

Consistent with Other Human Error Data

When most people look at Tables 1 2, and 3, their first reaction is that such high error rates are impossible. In fact, they are not only possible. They are entirely consistent with data on human error rates from other work domains. The Human Error Website (Panko, 2005a) presents data from a number of empirical studies. Broadly speaking, when humans do simple mechanical tasks, such as typing, they make undetected errors in about 0.5% of all actions. When they do more complex logical activities, such as writing programs, the error rate rises to about 5%. These are not hard and fast numbers, because how finely one defines reported "action" will affect the error rate. However, the logical tasks used in these studies generally had about the same scope as the creation of a formula in a spreadsheet.

The most complete set of data on error rates comes from programming, which is at least a cousin of spreadsheet development. In programming, many firms practice code inspection on program modules. In code inspection, teams of inspectors first inspect the module individually and then meet as a group to go over the module again (Fagan, 1976). Significantly, there is a requirement to report the number of errors found during code inspection. This has resulted in the publication of data from literally thousands of code inspections (Panko, 2005a). The data from these studies shows strong convergence. Code inspection usually finds errors in about 5% of all program statements after the developer has finished building and checking the module (Panko, 2005a). While there is some variation from study to study, much of this variation appears to be do to differences in programming language, module difficulty, and, sadly, in hastiness in development.

Note that we have been talking about uncorrected errors. Humans make many errors when they work, but they also correct many. Correction rates are high for mechanical errors, but they are considerably lower for logic errors, and if we have an omission caused by a misdiagnosis of the situation, the correction rate is very low. Interestingly, data from a study by Rabbit and Vyas (Rabbit & Vyas, 1970) suggest that a great deal of error correction takes place below the level of awareness, so people are unaware of how many errors they make and how many they correct.

There is even an emerging theory for why we make so many errors. Reason (Reason, 1990) has presented the most complete framework for understanding why human beings err, but many other writers have added their own contributions (Baars, 1992). In general, the emerging theory argues that human beings are amazingly fast and flexible (Reason, 1990) and can juggle multiple tasks and constraints (Flower & Hayes, 1980). However, the same cognitive processes that allow them to work this way inevitably result in occasional errors. Overall, Alexander Pope wrote that "To err is human." Today, we can both explain and quantify that statement.

Measuring Errors

The field audit studies shown in Table 1 report a simple measure of spreadsheet errors—the percentage of spreadsheets that contain at least one serious error. (Minor errors are discounted). While this is a vivid statistic, it does not tell us how many errors an average spreadsheet contains.

In addition, we have long known that in tasks that contain many subtasks, error rates will multiply along cascades of subtasks . In spreadsheeting, for instance, many bottom-line values are computed through long cascades of numerical and formula cells. Lorge and Solomon (1955) gave several formulas for calculating bottom-line error rates. Equation 1 is adapted from one of their simpler formulas. Here, E is the bottom line error rate, e is the task error rate (assumed to be the same for all tasks), and n is the number of task steps.

Equation 1: $E = 1 - (1-e)^n$

In programming, this link between individual error rates and cascade error rates is addressed by computing the error rate per line of code. More specifically, programmers measure the number of faults per thousand lines of non-comment source code (faults/KLOC). This statement error rate is roughly independent of program size (Putnam & Myers, 1992). Although it is important to make adjustments for complexity, size, and other matters for fine analysis (Ferdinand, 1993), and while it would be better to measure faults per function point, faults/KLOC is a good rough measure of programming errors.

In spreadsheets, a similar metric is the cell error rate (CER). This is the number of errors divided by the combined number of numerical and formula cells. (Label cells are like comment cells, which are ignored in faults/KLOC). Equivalently, this is the percentage of non-label cells containing errors. Table 1 shows that just as faults/KLOC usually falls into a narrow range (Panko, 2005a), the CERs seen in spreadsheet studies have been very similar. The relatively few divergent numbers that appear within a life cycle stage, furthermore, come when only certain cells are considered—generally cells with high potentials for error.

Errors by Life Cycle Stage

In programming, error rates vary by life cycle stage. Programmers make many errors when they are building a module but catch many of these errors before they finish the module and submit it to code inspection, data testing, or both. Module code inspection and testing catch more errors, and a second wave of code inspection and testing at final assembly catches still other errors. Relatively few errors should survive into the final operational systems.

Errors During Cell Entry

Only two studies have looked at errors during cell entry (Lerch, 1988; Olson & Nilsen, 1987-1988), that is, when the developer is entering formulas. One more study looked at errors when the subjects were typing label cells (Floyd & Pyun, 1987). These data, while limited, confirm that people make many errors while they work and catch quite a few of them (but not all). In the methodologies used, subjects were interrupted by the experimenter to correct errors as errors were made, so we do not know final error correction rates from these studies.

One interesting pattern from the Lerch (1988) study is that mechanical error rates rose dramatically when equations contained references to cells that were in both different columns and different rows than the cell containing the formula.

Errors at the End of the Development Stage

Most studies in Table 2 looked at errors at the end of the development stage, when subjects said that they were finished developing their spreadsheets. Apart from research that looked only at selected high-risk formulas (Janvrin & Morrison, 1996 2000), cell error rates across these studies were similar, despite the fact that subjects ranged from novices to highly experienced spreadsheet developers. One study (Panko & Sprague, 1998) even compared undergraduate business students, MBA students with little spreadsheet developing experience, and MBA students with more than 250 hours of spreadsheet development experience. Their CERs were very similar. Even when a task (the Wall task) was selected to be very simple and almost completely free of domain knowledge requirements (Panko & Sprague, 1998; Teo & Tan, 1997), about 40% of all spreadsheets contained errors, and the CER was about 2%.

Errors Found and Missed in Code Inspection

Code inspection or some other form of intensive testing may be needed to detect errors that remain at the end of the development stage. Fagan (Fagan, 1976) developed a comprehensive discipline for code inspection. As noted earlier, this method has a group of individuals inspect a program module individually by going through the module line by line. Then, the team meets as a group and again goes through the module line by line. There are strong guidelines for things such as maximum length for the module being examined (usually 100 to 400 lines of code) and maximum inspection rates (usually 100 to 200 lines per hour). If these rates are exceeded, error detection rates fall considerably (Panko, 2005a).

Experience has shown that code inspection is very difficult. In experiments, it has been found that individual inspectors catch only half or fewer of all errors in seeded program modules (Basili & Selby, 1986; Johnson & Tjahjono, 1997; Myers, 1978). Even real-world team inspection usually only catches 80% or fewer of all errors in a program module (Panko, 2005a).

In code inspection, as in development, spreadsheeting has shown strong parallels to programming. In three experiments that involved individual subjects code inspecting

spreadsheets seeded with errors, subjects also caught about half of all errors or fewer (Galletta *et al*., 1993; Galletta, Hartzel, Johnson, & Joseph, 1997; Panko & Sprague, 1998). These studies, however, did not impose traditional time disciplines or a team code inspection phase.In one study (Panko, 1999), traditional time disciplines and a group inspection phase were used in a code inspection of a spreadsheet by undergraduate MIS majors. The individuals found 67% of all seeded errors in this study, and group code inspection brought the discovery rate to 92%. However the groups discovered no errors that had not been discovered during the individual code inspection phase. In fact, one group failed to record an error during the group phase that one of its members had caught during the individual phase.

Errors in Operational Spreadsheets

Laboratory experiments always raise questions in the minds of many people. Fortunately, we have data from code inspections of operational spreadsheets, beyond the Hicks study just described. These inspections lacked the rigorous approach used in formal programming code inspection, however, so they probably found only a fraction of the errors in the spreadsheets they examined.

The first study, by Davies and Ikin (Davies & Ikin, 1987), inspected 19 operational spreadsheets from 10 developers in 10 different firms. Subjects expressed confidence in the correctness of their spreadsheets. However four of the spreadsheets (21%) were found to have serious quantitative errors, and 76% had quantitative or qualitative errors. One error involved a $7 million funds transfer between divisions. In another case, there were inconsistent currency conversion numbers in different parts of the spreadsheet. A third problem was a negative balance for stock on hand. There was inadequate documentation in 68% of the spreadsheets. Ten of the 19 failed to use cell protection. Only one had used audit software to audit the spreadsheets, and manual audits were "rare." Unfortunately, the authors did not describe their inspection methodology.

Later, Cragg and King (Cragg & King, 1993) inspected 20 operational spreadsheets from 10 firms. This audit found serious errors in 5 of the spreadsheets (25%). This time, the authors described their methodology. A single person conducted two-hour code inspections. Because the spreadsheets ranged from 150 to 10,000 cells in size, the code inspection time was much shorter than code inspection disciplines would allow. The authors themselves noted that they probably caught only some of the errors in these spreadsheets. The authors noted that the spreadsheets were characterized by poor layout, the use of cell protection in only 30%, the use of range names in only 45%, and other design flaws. Only half had clearly separated sections for input and output, and all mixed calculation cells with output cells. Only two had checked formulas before entering them in the spreadsheet. Eighty percent had been checked with input data, although the depth of such checking varied. Only one had previously been examined by another person. Half used macros, indicating considerable complexity and sophistication.

One universal aspect of the Cragg and King (Cragg & King, 1993) spreadsheets was informal iterative development. None of the spreadsheets used formal specification, design, or coding. In addition, there was extensive revision because of poor initial design. There had been a median of seven revisions, despite the fact that the spreadsheets had only been in use a median of six months. In 17 of the spreadsheets, the structure of the spreadsheet had to be changed during revisions. In six cases, there had been problems with the spreadsheet in previous releases.

Later, in a personal communication with the author, Butler (1996) described the audit of 273 spreadsheets submitted to HM Customs and Excise tax agency in the United Kingdom. The audit methodology involved the use of a program designed to look for inconsistencies in the spreadsheet. If the program tagged a portion of the spreadsheet, an individual auditor examined that portion of the spreadsheet for errors. This program probably caught only a fraction of all errors, because it could not catch such things as omissions, numbers being mistyped, most incorrect formulas, and many other types of errors. Nevertheless, the team found errors in 10.7% of the spreadsheets.

Since 1995, five studies have examined spreadsheets using better methodology. Hicks (1995) audited an enormous capital budgeting spreadsheet with 3,856 cells. This was a three-person, cell-by-cell code inspection. Errors were found in only 1.2% of all lines of code, but the errors would have been over a billion dollars if they had not been caught.

In England, certain spreadsheets have to be audited by law. Both Coopers and Lybrand and KPMG have active spreadsheet auditing consultancies. Together, they found significant errors in 45 of the spreadsheets they audited. Coopers & Lybrand only counted errors if they made some bottom-line value off by at least 5%. KPMG only listed "major errors." In 2003, the author interviewed auditors from these two organizations separately. Both said that they had never seen an error-free spreadsheet during audits. In addition, both independently gave data indicating that about 5% of all spreadsheets they audited have had "very serious errors." They noted that this percentage may be low for other types of spreadsheets because the spreadsheets being audited have a well-defined format. In addition, both aconsultancies only use single-person code inspection.

In 2000, Ray Butler, who audited spreadsheets for excise tax collection in the UK, found actionable errors in six of seven spreadsheets audited using single-person code inspection. Actionable errors included demands for the payment of additional taxes or providing refunds. (In his earlier study, Butler used a less rigorous method designed to minimize the cost of auditing and only found errors in 10.7% of the audited spreadsheets).

Most recently, Lawrence and Lee (2004) in Australia audited 30 spreadsheets created to justify project financing. These audits used single-person code inspection. The spreadsheets averaged 2,182 formulas, and on average 6.9% of the formula cells had auditing issues. It took an average of six iterations before the spreadsheets could be signed off.

Overall, what we see in operational spreadsheets is consistent with data from laboratory studies. Although laboratory studies generally have higher error rates, field studies did not use methodologies that probably could find most errors. In the four field audits for which we have cell error rates, furthermore, the CERs, which range from 0.4% to 6.9%, are similar in magnitude to CERs seen in experiments. In addition, most studies used single-person code inspection, which misses some errors.

Does Groupwork Help?

In their ethnographic study of 11 spreadsheet developers, Nardi and Miller (Nardi & Miller, 1991) noted that all developers used at least one other person during development. In some cases, the other person helped the developer code difficult parts of the spreadsheet. In other cases, two or more people used the spreadsheet development process as a way to share domain knowledge. In other cases, the other person examined input numbers and results for reasonableness, thereby helping with error detection. Nardi and Miller called this last pattern "cooperative debugging."

This led Panko and Halverson (Panko & Halverson, 1997) to examine group spreadsheet development. Their subjects developed a pro forma income statement from a word problem, working alone, in groups of two (dyads), or in groups of four (tetrads). Dyads reduced errors by about a third, while tetrads reduced errors by about two thirds. Only the latter was statistically significant. Furthermore, group development was only effective for catching certain types of errors.

The original Panko and Halverson (Panko & Halverson, 1997) study allowed some subjects to work outside the laboratory, in order to avoid the artifacts of having to work under contrived conditions. Of course this also lost experimental control. In a1999, Panko and Halverson redid the study in the laboratory, using individual and triadic development. They also reworded their task to remove ambiguities and one portion that subjects in the first study found extremely difficult. Taking these changes into account, the new study gave almost identical results.

Finally, as noted earlier, a study by the author (1999) found that team code inspection allowed undergraduate MIS majors to find 83% of all seeded errors in a spreadsheet, although the group did not find errors not previously found by the members of the team, who had inspected it alone before the group code inspection.

Overall, group development and testing appear to be promising areas to pursue.

Types of Errors

When many people think of spreadsheet errors, they think of someone mistyping a number, typing a plus sign for a minus sign in a formula, or pointing to the wrong cell when entering a formula. While these errors do exist, there are many other types of errors in spreadsheet development. Panko and Halverson (Panko & Halverson, 1996) give a taxonomy of error types.

First, there are quantitative errors, in which the spreadsheet gives an incorrect result. The studies in Table 1 look only at quantitative errors. However there are also qualitative errors that may lead to quantitative errors later, during maintenance, what-if analysis, or other activities. Reason (Reason, 1990) uses the term "latent errors" for such problems. Teo and Tan (Teo & Tan, 1997) demonstrated in an experiment how one type of qualitative error led to quantitative errors during what-if analysis.

Panko and Halverson (Panko & Halverson, 1996), following Allwood (Allwood, 1984) also found it useful to distinguish between three types of quantitative errors. Mechanical errors are simple mistakes, such as mistyping a number or pointing to the wrong cell. Logic errors involve entering the wrong formula because of a mistake in reasoning. As noted earlier, logic error rates are higher than mechanical error rates. Logic errors also more difficult to detect and correct (Allwood, 1984). The most dangerous type of error is the omission error, in which something is left out. Omission errors appear to be extremely difficult to detect (Allwood, 1984; Bagnara, Stablum, Rizzo, Fontana, & Ruo, 1987; Woods, 1984).

When Panko and Halverson (Panko & Halverson, 1997) analyzed the types of errors that subjects made while developing a spreadsheet, they found that all three forms of errors were common. Later, Panko and Sprague (Panko & Sprague, 1998) found the same broad pattern of errors. Panko and Halverson compared different types of errors to multiple lethal poisons. Even if all errors of the other two types were eliminated, each type of error alone would have produced an unacceptable number of incorrect spreadsheets.

Techniques to reduce errors may be better at finding some types of errors than others, so it is important to begin developing corpuses (collections) of errors to learn how frequent different types of errors really are.

Development Practices and Policies

Developer Surveys

A few studies have examined the processes that developers use to create spreadsheets. These studies asked the developer to select a single spreadsheet and describe it in detail. This method presumably resulted in bias toward larger-than-average spreadsheets. Even with this caveat, however, the results have been extremely interesting.

The Davies and Ikin (Davies & Ikin, 1987) and Cragg and King (Cragg & King, 1993) field audits mentioned earlier questioned developers about their spreadsheet development practices. Both found very informal development processes, little systematic code inspection, and little use of many other important development disciplines. Cragg and King noted that half their spreadsheets had been built without prior design or planning, almost half had poor design, half had poor layout, and only half had documentation of any type. One of their subjects commented that he or she had trouble understanding old spreadsheets.

Earlier, we mentioned the ethnographic interviews of Nardi and Miller (Nardi & Miller, 1991). In general, they found that developers took considerable precautions developing their spreadsheets, although the interviews did not gather quantitative data. The respondents were keenly aware of the dangers of error and spent considerable time working to reduce errors, including the creation of cross-footings, doing spot checks of formulas with calculators, and having others look at output. However only one of the 11 interviewees said that they had done a complete code inspection of a spreadsheet; this was a spreadsheet given to her by someone else (Nardi, 1993). As noted above, some had others look at their results, because errors could become invisible to the author through over-familiarity. However this falls far short of code inspection.

Later, Hendry and Green (Hendry & Green, 1994) did a similar ethnographic study, this time with 10 spreadsheet developers. They added a phase in which the subject walked through a spreadsheet with an author. They found that subjects often had a difficult time explaining parts of the spreadsheet that they had themselves built. They also found that some of the developers had run into serious problems trying to make parts of the spreadsheet analysis fit the row-and-column layout of spreadsheet program. The subjects found debugging to be particularly difficult. As a consequence of such difficulties, the subjects did a number of things to anticipate errors. Retrospectively in communication with the author, Hendry (1994) said that only three of the developers were "highly numerate" and did many things to avoid errors. In another personal communication with the author, Green (1994) said that detailed code inspection did not appear to be part of the spreadsheet development "culture."

Overall, both Nardi and Miller (Nardi & Miller, 1991) and Hendry and Green (Hendry & Green, 1994) found that subjects took considerable pains when they built spreadsheets.

Yet given the problems that users encountered, which were especially evident with the Hendry and Green methodology, it is unclear how successful user error-reduction errors are. Given the infrequency of comprehensive testing, however, complete success seems unlikely.

Other studies have used questionnaire surveys to collect data from larger numbers of respondents. For instance, Schultheis and Sumner (Schultheis & Sumner, 1994) had 32 MBA students each discuss their most recent spreadsheet. Sizes for the spreadsheets were not given, and a quarter of the respondents had developed five or fewer spreadsheets. The study had respondents rate each spreadsheet on 11 clusters of risks and the use of 9 clusters of control practices during development. Each cluster had several listed risks or control practices. The most frequent risks cited by subjects included the life expectancy of the application (which may exceed the tenure of the developer at the firm), use beyond the developer within firm, use beyond the firm, the number of users, and the level of complexity of the spreadsheet.

Higher-risk spreadsheets used slightly more controls (6.7) than lower-risk spreadsheets (4.6), but the use of controls was low in both cases. The most widespread control was verification of the logic, but this involved a grab bag of approaches of different sophistication. Documentation ranked third, after participation in formal training, but "there was very little documentation," with a quarter of the spreadsheets having none at all. Formal audits and formal reviews were among the least-used controls. Of the 32 spreadsheets, only five were reviewed by an auditor or consultant, and only ten had their designs reviewed by anyone other than the developer.

On a larger scale, Floyd, Walls, and Marr (Floyd, Walls, & Marr, 1995) sent questionnaires to 72 end users in four corporations. Each was asked to describe a single spreadsheet. The average size of the selected spreadsheets was 6,000 cells. The subjects said that their spreadsheets were important in the sense that important decisions were based on them. They also said, although to a lesser extent, that their spreadsheets had a material financial impact on the firm. Materiality, by the way, was not correlated with size, and "small and large spreadsheets were equally likely to support material decisions." (41). Eighty percent said that they would be able to work without the spreadsheet, although with some difficulty. One in six, however, said they would not be able to work without the spreadsheet. Development was strongly iterative. There was a high level of confidence that the spreadsheet described was correct.

Hall (Hall, 1996) surveyed 106 spreadsheet developers in Australia. The average size of the selected spreadsheets was 216 KB, and because most spreadsheets were developed using Lotus 1-2-3, this implied considerable size. Among her findings were the following:

1. The spreadsheets were important. Only 7% of the spreadsheets were of low importance; 39% were of high importance. 27% modified existing corporate data; 49% created new corporate data. 67% were run on a regular basis, 16% occasionally, and only 17% once or a few times. Only 17% had results used only

by the developer, while 23% had results used by many departments in the organization, and 29% had results used outside the firm.
2. The spreadsheets were complex. 45% used macros. 36% had links to other spreadsheets. 21% had links to databases. 47% used IF functions. 66% used absolute referencing.
3. Development used only limited safeguards. Only half had modular designs. Only 49% used cell protection. Only 46% used cross-footing.
4. Data testing was limited. 71% checked formulas with data, but only 50% did so with results determined ahead of time, only 42% used test data with errors, and only 33% used test data at the limits of the normal range.
5. Inspection was limited. Only 17% were checked by another developer, 5% by an internal auditor, and 6% by an external auditor (there was overlap in these percentages).
6. 18% said that their project had been rushed.

Interestingly, Hall asked respondents if they used a number of controls and whether they should have used these controls. For almost all controls, more developers acknowledged that they should have used it than said that they actually did use it. In general, experienced developers were more likely to use most controls than inexperienced developers, but the difference was only modest.

Overall, these studies show that many spreadsheets are large, complex, important, and affect many people. Yet development tends to be quite informal, and even trivial controls such as cell protection are not used in most cases. In programming, code inspection and data testing are needed to reduce error rates after a module is developed. Yet code inspection is very infrequent, and while data testing is done, it lacks such rigors as the use of out-of-bounds data. In general, end user development in spreadsheeting seems to resemble programming practice in the 1950s and 1960s.

Corporate Policies

One reason why individual spreadsheet developers do not use rigorous development disciplines may be the fact that few organizations have policies on end user development in general, much less spreadsheet development.

Galletta and Hufnagel (Galletta & Hufnagel, 1992) surveyed 107 MIS executives using a mail questionnaire that focused broadly on policies for controlling end user development. For restrictions on end user development, 23% said they had rules and 58% said they had guidelines. Yet even for corporations with rules and guidelines, the respondents said that there was full compliance only 27% of the time. When asked about requirements for post-development audits, only 15% said they had rules, and another 34% said they had guidelines. The respondents said that where they had rules and guidelines for post-development audits, furthermore, there was full compliance only 10% of the time, and guidance was completely ignored 41% of the time.

Cale (Cale, 1994) surveyed 52 IS and non-IS managers in 25 firms. This questionnaire focused on testing and documentation. Cale found that only 10% of the companies had written policies for testing, while about another quarter had "unwritten policies." Documentation percentage were similar. About 70% strongly agreed that this lack of testing standards was producing serious problems. None disagreed. About 90% would impose testing standards for spreadsheets taking a week or more to develop, if the spreadsheet were used by multiple departments, or if the output would update databases in the developer's department or other departments.

Floyd, Walls, and Marr (Floyd, *et al*., 1995), in their survey mentioned earlier, found that only about 15% of the firms had development policies, 40% implementation policies, and two-thirds development modification policies. Those whose spreadsheets were rated as more important than the average did not have a fuller set of policies for development. None of the respondents reported a comprehensive set of policies for all phases of development. Furthermore, the policies that did exist were created and implemented at the workgroup level. Floyd, Walls, and Marr referred to the workgroup-centric nature of policies a "clan-based" social policy based primarily on socialization. No respondent, however, knew of a spreadsheet disaster in their firm.

Speier and Brown (Speier & Brown, 1996) surveyed 22 members of three departments about end user control policies in general. The company had few corporate-wide rules except for backup rules, which were not enforced anyway. Most "rules" were informal and departmental and differed widely across departments and within the perceptions of different users in each department. This reinforces the clan-based control concept of Floyd, Walls, and Marr (Floyd *et al*., 1995).

Finally, Hall (Hall, 1996), in her survey mentioned earlier, asked respondents about corporate policies. Only 11% of the respondents claimed to know of a comprehensive corporate policy for spreadsheet development. Of these, furthermore, only one in three answered "Yes" when asked if they knew where to find the policy in written form! In 90% of the cases where rules existed, furthermore, enforcement was left to the developer. Interestingly, 39% said that they had complied with their organization's policy—four times the percentage saying that such a policy existed. Perhaps these respondents felt that they were complying with unwritten rules.

Overall, formal spreadsheet policies seem to be fairly rare, and their enforcement seems to be casual. While more respondents seem to believe that informal and typically unwritten guidelines exist, the effectiveness of such guidelines can only be left to conjecture.

Overconfidence

Why is end user development in spreadsheeting so casual? The answer may be that both developers and corporations are overconfident about the accuracy of spreadsheets, despite the large amount of information to the contrary. As noted earlier,

when Brown and Gould (Brown & Gould, 1987) gave three spreadsheet development tasks to nine highly experienced spreadsheet developers, all made at least one error, and 63% of the spreadsheets overall had errors. Yet when asked about their confidence in the correctness of their spreadsheets, their median score was "very confident." Davies and Ikin (Davies & Ikin, 1987), who found errors in four of the 19 spreadsheets they audited, also found that the developers were extremely confident in the accuracy of their spreadsheets. As noted above, Floyd, Wall, and Marr (Floyd & Pyun, 1987) found that their 72 end users were confident in the correctness of their spreadsheets, despite their large sizes (averaging 6,000 cells).

In larger spreadsheets, there should be a higher probability of an error than in a small spreadsheet. Yet when Reithel, Nichols, and Robinson (Reithel, Nichols, & Robinson, 1996) had subjects examine spreadsheets and then state their confidence in the spreadsheet, subjects rated large well-formatted spreadsheets much higher than large plainly-formatted spreadsheets and also higher than small spreadsheets regardless of formatting. In a personal communication with the author in 1997, Reithel noted that for the large fancy spreadsheet, 72% of the subjects gave the spreadsheet the highest confidence range (80%-100%), and only 10% chose the 1%-60% range. For the other three conditions, about a third of the respondents chose these two ranges. Unfortunately, the reasons for this apparently inverted pattern of confidence are unclear.

In experiments, Panko and Halverson (1997) and Panko and Sprague (1998) found high confidence among subjects despite the fact that these subjects made many errors. In both studies, furthermore, confidence in correctness was unrelated to the actual number of errors. More recently, Panko and Halverson (2001) had subjects develop a spreadsheet individually or in triads (groups of three). When asked to estimated the probability that their spreadsheet contained an error, individuals had a mean of 18%, while members of triads had a mean of 13%. The actual figures percentage of incorrect spreadsheets 86% and 27% for individuals and triads.

While such massive levels of overconfidence seem unreasonable, overconfidence is perhaps the most well-established phenomenon in the behavioral sciences. When asked about many things, such as driving, a large majority of all people rate themselves as above average (Brown, 1990; Gilovich, 1991; Koriat, Lichtenstein, & Fischoff, 1980). A long stream of experiments have found that people are overconfident in a wide variety of tasks ranging from Trivial Pursuit type questions to the perception of acoustical signals (Adams & A., 1961; Lichtenstein, Fischoff, & Philips, 1982; Plous, 1993; Wagenaar & Keren, 1986). Overconfidence has also been seen in decision support systems use (Kasper, 1996) and database queries (Greenblatt & Waxman, 1978). Even experts in a wide variety of fields are often appallingly overconfident within their domains of expertise (Johnson, 1988; Shanteau & Phelps, 1977). Overall, in both contrived and realistic tasks, there seems to be a pervasive human tendency toward overconfidence.

One of the most interesting aspects of this overconfidence is the hard-easy effect, in which overconfidence is highest when actual performance is lowest (Clarke, 1960;

Dunning, Griffin, Milojkovic, & Ross, 1990; Lichtenstein *et al.*, 1982; Plous, 1993; Wagenaar & Keren, 1986). In practice, as degree of difficulty increases, confidence falls, but not proportionately.

Overconfidence is corrosive because it tends to blind people to the need for taking steps to reduce risks. In driving, for instance, there is evidence that people act as if they were almost blind to low-probability but lethal dangers (Howarth, 1988; Swensen, 1977). In fact, when we take unwarranted driving risks, our risky behavior is reinforced, because we get to out destinations faster, do less work, and so forth (Fuller, 1990). Even if we get into trouble and have to take evasive action, we tend to interpret this as validation of our skills rather than foolishness (Fuller, 1990; Habberley, Shaddick, & Taylor, 1986). In spreadsheeting, in turn, if we do not do formal testing and follow other tedious disciplines, we benefit by saving time and avoiding onerous work. When we do catch errors informally, furthermore, we become convinced of our efficacy in error correction.

More generically, when we search for errors, we seem to have three basic processes (Allwood, 1984), two of these appear to be triggered by mismatches between the state of the world and what we had expected it to be. The third, the "standard check," involves checking for errors even when we have no direct suspicion that they exist. In other words, the standard check is like code inspection, although it may be for only part of the spreadsheet. The standard check is important because although it is not motivated except by self-discipline, it appears to be the only way to catch complex logical errors (Allwood, 1984; Rizzo, Bagnara, & Visciola, 1986).

How much standard error checking should we do? Obviously, it would be insane to question everything we do in life (Gilovich, 1991). We must have some "stopping rules," and those stopping rules will determine what we find (Rasmussen, 1990). If we are overconfident, our standard error checking is likely to stop too soon.

Overconfidence tends to be greatest when real-world subjects do not get detailed feedback over many cases (Shanteau & Phelps, 1977; Wagenaar & Keren, 1986). In experiments, (Arkes, Christensen, Lai, & Blumer, 1987; Lichtenstein & Fischoff, 1980), too, detailed feedback has been effective. Note, however, that experience is not enough. There must be very detailed feedback on each case. For instance, Wagenaar and Keren (Wagenaar & Keren, 1986) found that expert bridge players, who received detailed feedback after each hand and analyzed it, were well calibrated in confidence. In contrast, professional blackjack dealers, who see but do not analyze the results of each deal, were no better calibrated than lay people. In spreadsheeting, we have seen that detailed code inspection is rare, so few developers are likely to be getting the specific feedback they need to reduce their natural overconfidence.

Implications for Development

Although we still have far too little knowledge of spreadsheet errors to come up with a definitive list of ways to reduce errors, the similarity of spreadsheet errors to programming errors suggests that, in general, we will have to begin adopting (yet adapting) many traditional programming disciplines to spreadsheeting. In effect, we must teach new dogs old tricks. See, for example, the paper by Rajalingham, Chadwick, Knight, and Edwards.

One aspect noted in past studies (Brown & Gould, 1987; Panko & Halverson, 1997) is the tendency for spreadsheet developers to begin building without an adequate preliminary design. In large spreadsheets built with many iterations, this is very likely to lead to problems. Benham, Delaney, and Luzi (Benham, Delaney, & Luzi, 1993), Ronen, Palley, and Lucas (Ronen, Palley, & Lucas, 1989), and Isakowitz, Schocken, and Lucas (Isakowitz, Schocken, & Lucas, 1995) have even suggested the imposition of fairly formal planning techniques using CASE-like tools. However in spreadsheet development, building the spreadsheet typically results in incrementally greater understanding of the problem, so it may be unreasonable to expect full prior design. We many need something more like plans in writing, which give guidance yet should be kept "cheap enough" to be thrown away when obsolete (Hayes & Flower, 1980).

Most prescriptions for spreadsheet development have focused on late design and development phase, including modular construction, having an assumptions section for all inputs, and using cell protection. These are certainly important, but they are far from being enough. If people do and feel good about such things, in fact, they may become overconfident. In addition, assumptions sections, while desirable in many ways, may actually create errors. The experiments of Janvrin and Morrison (Janvrin & Morrison, 1996 2000) and Lerch (Lerch, 1988) suggest that people make many errors when they point to distant cells, as is the case when as assumption section is used. Hall (Hall, 1996) lists many other development-phase controls.

In programming, a new form of development is called cleanroom development (Linger, 1994). In this approach, the program is proven to be correct mathematically before it is written. In spreadsheeting, our experience observing developers is that they often enter a formula without thinking it through beforehand. Unless the results look obviously wrong, they do not try to determine if the formula is incorrect. This allows the survival of many subtle yet important errors, even when the developer really knows approximately what the correct answer should be. In many ways, the developers seem to be acting like Thorndike's (Thorndike, 1898) cats, who upon being put in a puzzle box simply tried things until they got out (for a modern discussion, see Mayer (Mayer, 1992)). One approach to reducing such behavior is to require developers to write out and critique all non-trivial formulas, doing calculations by hand, and then replicating these calculations with the formula typed into the spreadsheet.

Most prescriptive papers also discuss the importance of documentation. As we have already seen, documentation is rare in spreadsheets.

In programming, we have seen from literally thousands of studies that programs will have errors in about 5% of their lines when the developer believes that he or she is finished (Panko, 2005a). A very rigorous testing stage after the development stage is needed to reduce error rates by about 80% (Panko, 2005a). Whether this is done by data testing, line-by-line code inspection, or both, testing is an onerous task and is difficult to do properly. In code inspection, for instance, we know that the inspection must be done by teams rather than individuals and that there must be sharp limits for module size and for how many lines can be inspected per hour (Fagan, 1976). We have seen above that most developers ignore this crucial phase. Yet unless organizations and individuals are willing to impose the requirement for code inspection or comprehensive data testing, there seems little prospect for having correct spreadsheets in organizations. In comparison, the error reduction techniques normally discussed in prescriptive spreadsheet articles probably have far lower impact.

Whatever specific techniques are used, one broad policy must be the shielding of spreadsheeters who err from punishment. In programming, it has long been known that it is critical to avoid blaming in code inspection and other public processes. For instance, Fagan (Fagan, 1976) emphasized that code inspection should never be used for performance evaluation. As Beizer (Beizer, 1990) has emphasized, a climate of blaming will prevent developers from acknowledging errors. Edmondson (Edmondson, 1996) found that in nursing, too, a punitive environment will discourage the reporting of errors. Quite simply, although the error rates seen in research studies are appalling, they are also in line with the normal accuracy limits of human information processing. We cannot punish people for normal human failings.

Finally, we have been discussing developers in this paper, but Dhebar (Dhebar, 1993) has emphasized the need for employees in general to be smart consumers of spreadsheet results generated by others. In universities, introductory classes deal with both students who will probably be both developers and consumers of spreadsheets. Our class content should reflect both roles.

Although many prescriptive techniques have been put forth, only code inspection has been tested experimentally and has proven to be both safe and effective. Individual code inspection has been shown to find about half of all errors, so that group code inspection is needed. As Panko (1999) has shown, even group inspection helps only somewhat, although it helps especially for difficult-to-find errors. Other prescriptive methods have been put forth but have not been tested and so must be viewed somewhat skeptically. For instance, in unpublished results found by the author, when students who put assumptions sections at the start of their spreadsheet were compared with students who did not use assumption sections, there was no statistically significant difference in error rates, and students who did not have assumptions sections actually had numerically *lower* error rates.

Conclusion

All in all, the research done to date in spreadsheet development presents a very disturbing picture. Every study that has attempted to measure errors, without exception, has found them at rates that would be unacceptable in any organization. These error rates, furthermore, are completely consistent with error rates found in other human activities. With such high cell error rates, most large spreadsheets will have multiple errors, and even relatively small "scratch pad" spreadsheets will have a significant probability of error.

Despite the evidence, individual developers and organizations appear to be in a state of denial. They do not regularly implement even fairly simple controls to reduce errors, much less such bitter pills as comprehensive code inspection. One corporate officer probably summarized the situation by saying that he agreed with the error rate numbers but felt that comprehensive code inspection is simply impractical. In other words, he was saying that the company should continue to base critical decisions on bad numbers.

A major impediment to implementing adequate disciplines, of course, is that few spreadsheet developers have spreadsheeting in their job descriptions at all, and very few do spreadsheet development as their main task. In addition, because spreadsheet development is so dispersed, the implementation of policies has to be left to individual department managers. While organizations might identify critical spreadsheets and only impose hard disciplines on them (Panko, 1988), this would still mean that many corporate decisions would continue to be made on the basis of questionable analyses.

References

Adams, J. K., & A., A. P. (1961). Realism of Confidence Judgements. *Psychological Review, 68*, 35-45.

Allwood, C. M. (1984). Error Detection Processes in Statistical Problem Solving. *Cognitive Science, 8(4),* 413-437.

Arkes, H. R., Christensen, C., Lai, C., & Blumer, C. (1987). Two Methods of Reducing Overconfidence. *Organizational Behavior and Human Decision Processes, 39,* 134-144.

Baars, B. J. (1992). A New Ideomotor Theory of Voluntary Control. In B. J. Baars (Ed.), *Experimental Slips and Human Error* (pp. 93-120). New York: Plenum.

Bagnara, S., Stablum, F., Rizzo, A., Fontana, A., & Ruo, M. (1987,). Error Detection and Correction: A Study on Human-Computer Interaction in a Hot Strip Mill Planning and Production System. *Preprints of the First European Meeting on Cognitive Engineering Approaches to Process Control*, Marcoussis, France.

Basili, V. R., & Selby, R. W., Jr. (1986). Four Applications of a Software Data Collection and Analysis Methodology. In J. K. Skwirzynski (Ed.), *Software System Design Methdology* (pp. 3-33). Berlin: Springer-Verlag.

Beizer, B. (1990). *Software Testing Techniques*. (2nd ed.). New York: Van Nostrand.

Benham, H., Delaney, M., & Luzi, A. (1993). Structured Techniques for Successful End User Spreadsheets. *Journal of End User Computing, 5(2),* 18-25.

Brown, I. D. (1990). Drivers' Margins of Safety Considered as a Focus for Research on Error. *Ergonomics, 33(10/11),* 1307-1314.

Brown, P. S., & Gould, J. D. (1987). An Experimental Study of People Creating Spreadsheets. *ACM Transactions on Office Information Systems, 5(3),* 258-272.

Butler, R. J., "Is this Spreadsheet a Tax Evader? How H. M. Customs & Excise Tax Test Spreadsheet Applications," *Proceedings of the 33rd Hawaii International Conference on System Sciences*, Maui, Hawaii, January 4-7, 2000.

Butler, Raymond, Tax audit study in 1992, personal communications by electronic mail, August and September, 1996 and in August, 1997.

Cale, E. G., Jr. (1994). Quality Issues for End-User Developed Software. *Journal of Systems Management, 45(1),* 36-39.

Clermont, M., Hanin, C. & Mittermeier, R. (2000, July). A spreadsheet auditing tool evaluated in an industrial context. In *Proceedings of the EuSpRIG 2000 Symposium, Spreadsheet Risks—the Hidden Corporate Gamble*. (pp. 35-46). Greenwich, England: Greenwich University Press.

Clarke, F. R. (1960). Confidence Ratings, Second-Choice Responses, and Confusion Matrices in Intelligibility Tests. *Journal of the Acoustical Society of America, 32,* 35-46.

Coopers & Lybrand in London. Description available at http://www.planningobjects.com/jungle1.htm. Contact information is available at that webpage.

Cragg, P. G., & King, M. (1993). Spreadsheet Modelling Abuse: An Opportunity for OR? *Journal of the Operational Research Society, 44(8),* 743-752.

Davies, N., & Ikin, C. (1987). Auditing Spreadsheets. *Australian Account,* 54-56.

Davis, G. B. (January 1982). Caution: User-Developed Systems Can be Hazardous to Your Organization. *Proceedings of the Seventeenth Annual Hawaii International Conference on System Sciences,* Honolulu, Hawaii.

Dhebar, A. (1993). Managing the Quality of Quantitative Analysis. *Sloan Management Review, 34(2),* 69-75.

Ditlea, S. (1987). Spreadsheets Can be Hazardous to Your Health. *Personal Computing,* 60-69.

Dunning, D., Griffin, D. W., Milojkovic, J. D., & Ross, L. (1990). The Overconfidence Effect in Social Prediction. *Journal of Personality and Social Psychology, 4,* 568-581.

Edmondson, A. C. (1996). Learning from Mistakes is Easier Said than Done: Group and Organizational Influences on the Detection and Correction of Human Error. *Journal of Applied Behavioral Science, 32(1),* 5-28.

Fagan, M. E. (1976). Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal, 15(3),* 182-211.

Ferdinand, A. E. (1993). *Systems, Software, and Quality Engineering: Applying Defect Behavior Theory to Programming.* New York: Van Nostrand Reinhold.

Flower, L. A., & Hayes, J. R. (1980). The Dynamics of Composing: Making Plans and Juggling Constraints. In L. W. Gregg & E. R. Steinberg (Eds.), *Cognitive Processes in Writing* (pp. 31-50). Hillsdale, NJ: Lawrence Erlbaum Associates.

Floyd, B. D., & Pyun, J. (1987, October). *Errors in Spreadsheet Use* (Working Paper 167). New York: Center for Research on Information Systems, Information Systems Department, New York University.

Floyd, B. D., Walls, J., & Marr, K. (1995). Managing Spreadsheet Model Development. *Journal of Systems Management, 46(1),* 38-43, 68.

Freeman, D. (1996). How to Make Spreadsheets Error-Proof. *Journal of Accountancy*, *181(5),* 75-77.

Fuller, R. (1990). Learning to Make Errors: Evidence from a Driving Simulation. *Ergonomics, 33(10/11),* 1241-1250.

Galletta, D. F., Abraham, D., El Louadi, M., Lekse, W., Pollailis, Y. A., & Sampler, J. L. (1993). An Empirical Study of Spreadsheet Error Performance. *Journal of Accounting, Management, and Information Technology, 5(3-4),* 79-95.

Galletta, D. F., Hartzel, K. S., Johnson, S., and Joseph, J. L. (1997, Winter). "Spreadsheet Presentation and Error Detection: An Experimental Study" *Journal of Management Information Systems* (13:2), pp. 45-63.

Galletta, D. F., & Hufnagel, E. M. (1992). A Model of End-User Computing Policy: Context, Process, Content and Compliance. *Information and Management, 22(1),* 1-28.

Gilovich, T. (1991). *How We Know What Isn't So: The Fallibility of Human Reason in Everyday Life*. New York: Free Press.

Greenblatt, D., & Waxman, J. (1978). A Study of Three Database Query Languages. In B. Schneiderman (Ed.), *Databases: Improving Usability and Responsiveness* (pp. 77-97). New York: Academic Press.

Habberley, J. S., Shaddick, C. A., & Taylor, D. H. (1986). *A Behavioural Study of the Collision Avoidance Task in Bridge Watchkeeping*. Southhampton, England: College of Marine Studies.

Hall, M. J. J. (1996). A Risk and Control Oriented Study of the Practices of Spreadsheet Application Developers. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Kihei, Maui.

Hayes, J. R., & Flower, L. S. (1980). Identifying the Organization of Writing Processes. In L. Gregg & E. Steinbert (Eds.), *Cognitive Processes in Writing* (pp. 3-30). Hillsdale, NJ: Lawrence Erlbaum Associates.

Hendry, D. G., & Green, T. R. G. (1994). Creating, Comprehending, and Explaining Spreadsheets: A Cognitive Interpretation of What Discretionary Users Think of the

Spreadsheet Model. *International Journal of Human-Computer Studies, 40(6),* 1033-1065.

Hicks, L., NYNEX, personal communication with the first author via electronic mail, June 21, 1995.

Howarth, C. I. (1988). The Relationship Between Objective Risk, Subjective Risk, and Behaviour. *Ergonomics, 31,* 527-535.

Isakowitz, T., Schocken, S., & Lucas, H. C. J. (1995). Toward a Logical/Physical Theory of Spreadsheet Modeling. *ACM Transactions on Information Systems, 13(1),* 1-37.

Janvrin, D., & Morrison, J. (2000a). Factors Influencing Risks and Outcomes in End-User Development. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Kihei, Maui, Hawaii.

Janvrin, D., & Morrison, J. (2000b). Using a Structured Design Approach to Reduce Risks in End User Spreadsheet Development, *Information & Management, (37),* 1-12.

Johnson, E. J. (1988). Expertise and Decision Under Uncertainty: Performance and Process. In M. T. H. Chi, R. Glaser, & M. J. Farr (Eds.), The Nature of Expertise (pp. 209-228). Hillsdale, NJ: Lawrence Erlbaum Associates.

Johnson, P., & Tjahjono, D. (1997, May). Exploring the Effectiveness of Formal Technical Review Factors with CSRS. *Paper presented at the Proceedings of the 1997 International Conference on Software Engineering,* Boston, MA.

Kasper, G. M. (1996). A Theory of Decision Support System Design for User Calibration. *Information Systems Research, 7(2),* 215-232.

Koriat, A., Lichtenstein, S., & Fischoff, B. (1980). Reasons for Confidence. *Journal of Experimental Psychology: Human Learning and Memory, 6,* 107-117.

KPMG Management Consulting, "Supporting the Decision Maker - A Guide to the Value of Business Modeling," press release, July 30, 1998. http://www.kpmg.co.uk/uk/services/manage/press/970605a.html.

Lawrence, R. J. and Lee, J, "Financial Modelling of Project Financing Transactions," Institute of Actuaries of Australia Financial Services Forum, August 26-27 2004.

Lerch, F. J. (1988). *Computerized Financial Planning: Discovering Cognitive Difficulties in Knowledge Building.* Unpublished Ph.D. Dissertation, University of Michigan, Ann Arbor, MI.

Lichtenstein, S., & Fischoff, B. (1980). Training for Calibration. *Organizational Behavior and Human Performance, 26,* 149-171.

Lichtenstein, S., Fischoff, B., & Philips, L. D. (1982). Calibration of Probabilities: The State of the Art to 1980. In D. Kahneman, P. Slovic, & A. Tversky (Eds.), *Judgment Under Uncertainty: Heuristics and Biases* (pp. 306-334). Cambridge, England: Cambridge University Press.

Linger, R. C. (1994). Cleanroom Process Model. *IEEE Softwere, 11(2),* 50-58.

Lorge, I., & Solomon, H. (1955). Two Models of Group Behavior in the Solution of Eureka-Type Problems. *Psychometrika, 20(2),* 139-148.

Lukasik, Todd, CPS, private communication with the author, August 10, 1998.

Mayer, R. E. (1992). Thinking, Problem Solving, Cognition. (2nd. ed.). New York: W. H. Freeman.

Myers, G. J. (1978). A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections. *Communications of the ACM, 21(9),* 760-768.

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA: MIT Press.

Nardi, B. A., & Miller, J. R. (1991). Twinkling Lights and Nested Loops: Distributed Problem Solving and Spreadsheet Development. *International Journal of Man-Machine Studies, 34(1),* 161-168.

Olson, J. R., & Nilsen, E. (1987-1988). Analysis of the Cognition Involved in Spreadsheet Interaction. *Human-Computer Interaction, 3(4),* 309-349.

Panko, R. R. (2005a). *Human Error Website* (http://www.cba.hawaii.edu/panko/papers/ss/humanerr.htm) . Honolulu, HI: University of Hawaii.

Panko, R. R. (2005b). Spreadsheet Research (SSR) Website (http://www.cba.hawaii.edu/panko/ssr/). Honolulu, Hawaii: University of Hawaii.

Panko, R. R. (1988). *End User Computing: Management, Applications, and Technology*, New York: Wiley.

Panko, R. R., "Two Corpuses of Spreadsheet Error," *Proceedings of the Thirty-Third Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2000. (.pdf format)

Panko, Raymond R., (1999, Fall) "Applying Code Inspection to Spreadsheet Testing," *Journal of Management Information Systems*, (16:2), 159-176.

Panko, R. R., and Halverson, R. P., Jr. " (2001, January) "An Experiment in Collaborative Spreadsheet Development," *Journal of the Association for Information Systems*.

Panko, R. R., and Halverson, R. P., Jr. " (1997, Spring). "Are Two Heads Better than One? (At Reducing Errors in Spreadsheet Modeling?" *Office Systems Research Journal* (15:1), pp. 21-32.

Panko, R. R., & Halverson, R. P., Jr. (1996, January). Spreadsheets on Trial: A Framework for Research on Spreadsheet Risks. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences,* Maui, Hawaii.

Panko, R. R., and Sprague, R. H. J. (1999). "Hitting the Wall: Errors in Developing and Code-Inspecting a 'Simple' Spreadsheet Model." *Decision Support Systems*, (22), pp. 337-353.

Plous, S. (1993). *The Psychology of Judgment and Decsion Making*. Philadelphia: Temple University Press.

Putnam, L. H., & Myers, W. (1992). Measures for Excellence: *Reliable Software on Time, on Budget*. Englewood Cliffs, NJ: Yourdon.

Rabbit, P. M. A., & Vyas, S. M. (1970). An Elementary Preliminary Taxonomy for Some Errors in Laboratory Choice RT Tasks. In A. F. Sanbders (Ed.), *Attention and Performance III* (pp. 56-76). Amsterdam: North Holland.

Rajalingham, Kamalasen; Chadwick, David; Knight, Brian; and Edwards, Dilwyn, (2000, January)"Quality Control in Spreadsheets: A Software Engineering-Based Approach to Spreadsheet Development," P*roceedings of the Thirty-Third Hawaii International Conference on System Sciences*, Maui, Hawaii. (.pdf format)

Rasmussen, J. (1990). The Role of Perception in Organizing Behavior. *Ergonomics, 33(10/11),* 1185-1199.

Reason, J. T. (1990). *Human Error*. Cambridge, UK: Cambridge University Press.

Reithel, B. J., Nichols, D. L., & Robinson, R. K. (1996). An Experimental Investigation of the Effects of Size, Format, and Errors on Spreadsheet Reliability Perception. *Journal of Computer Information Systems,* 54-64.

Rizzo, A., Bagnara, S., & Visciola, M. (1986). Human Error Detection Processes. In G. Mancini, D. Woods, & E. Hollnagel (Eds.), *Cognitive Engineering in Dynamic Worlds*. Ispra, Italy: CEC Joint Research Centre.

Ronen, B., Palley, M. A., & Lucas, H. (1989). Spreadsheet Analysis and Design. *Communications of the ACM, 32(1),* 84-92.

Schultheis, R., & Sumner, M. (1994). The Relationship of Application Risks to Application Controls: A Study of Microcomputer-Based Spreadsheet Applications. *Journal of End User Computing, 6(2),* 11-18.

Shanteau, J., & Phelps, R. H. (1977). Judgment and Swine: Approaches and Issues in Applied Judgement Analysis. In M. F. Kaplan & S. Schwartz (Eds.), *Human Judgment and Decision Processes in Applied Setting*. New York: Academic Press.

Speier, C., & Brown, C. V. (1996, January). Perceived Risks and Management Actions: Differences in End-User Application Development Across Functional Groups. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Science*, Maui, Hawaii.

Swensen, O. (1977). *Risks of Road Transportation from a Psychological Perspective: A Pilot Study (Project Risk Generation and Risk Assessment in a Social Perspective*, Report 3-77): Committee for Future-Oriented Research, Stockholm, Sweden.

Teo, T. S. H., & Tan, M. (1997,). Quantitative and Qualitative Errors in Spreadsheet Development. *Proceedings of the Thirtieth Hawaii International Conference on System Sciences*, Maui, Hawaii.

Thorndike, E. L. (1898). Animal Intelligence: An Experimental Study of the Associative Processes of Animals. *Psychological Monographs, 2(8).*

Wagenaar, W. A., & Keren, G. B. (1986). Does the Expert Know? The Reliability of Predictions and Confidence Ratings of Experts. In E. Hollnagel, G. Manici, & D. D. Woods (Eds.), *Intelligent Decision Support in Process Environments* (pp. 87-103). Berlin: Springer-Verlag.

Woods, D. D. (1984). Some Results on Operator Performance in Emergency Events. *Institute of Chemical Engineers Symposium Series, 90,* 21-31.