# What We Know About Spreadsheet Errors

Raymond R. Panko
University of Hawaii, USA

*Although some spreadsheets are small "scratch pad" applications, many are large and complex, and many mission-critical decisions depend on spreadsheet analyses. 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 error rates 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 that 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 post-development testing.*

Spreadsheet programs are often seen as tools for building small and simple "scratch pad" applications. Yet many spreadsheets are large and complex (Floyd, Walls, & Marr, 1995; Hall, 1996, 78) and many mission-critical organizational decisions are guided by such non-trivial spreadsheets.

Unfortunately, as we will discuss below, there is strong and mounting evidence that spreadsheet errors are widespread. Much of this evidence is available in at least summarized form at the Spreadsheet Research (SSR) website (Panko, 1997b). For instance, this website has data from four systematic field studies involving over 300 operational spreadsheets. It also has data from over a dozen experiments that have used over 1,000 subjects. Every one of these studies, without exception, has found errors at rates that any reasonable person would find to be unacceptable.

The SSR website also contains information from many other studies, including surveys of spreadsheet developers and surveys of organizational control practices for spreadsheet developer. The results show a pattern of considerable care in development yet far less care than professional programmers use when they work, despite strong similarities between spreadsheet and programming errors in both frequency and type.

## Error Rates

### Introduction

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, 1997b). In

Manuscript originally submitted July 14, 1997; Revised September 25, 1997; Accepted October 26, 1997 for publication.

most cases, either the firm that made the error was forced to disclose its mistake, or consultants familiar with the error revealed it.

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

### Consistent with Other Human Error Data

When most people look at Table 1, 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, 1997a) presents data from a number of empirical studies. Broadly speaking, when humans do simple mechanical tasks, such as typing, they generally make undetected errors in about 0.5% of all actions. When they do more complex logical activities similar in complexity to spreadsheet development, the error rate generally rises to about 5%. So if we saw lower error rates in either spreadsheet experiments or field audits, we would have to question the results.

The most complete set of data on error rates comes from programming, which is at least a cousin of spreadsheet development and has similar error rates. 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). Data from literally thousands of code inspections (Panko, 1997a) converges to an error rate of roughly 5% of all program statements after the developer has finished building and checking the module (Panko, 1997a).

There is even an emerging theory for why we make so many errors (Reason, 1990). In general, the emerging theory argues that human beings are amazingly fast and flexible and can juggle multiple tasks and constraints. However, the same cognitive processes that allow us to work this way contain tricks that inevitably produce occasional errors. Alexander Pope wrote that "To err is human." Today, we can both explain why this is so and can even quantify it.

### 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.

A better measure is the cell error rate (CER), which is the percentage of numerical and formula cells that contain errors. If experience with programming is correct, CER should be roughly independent of spreadsheet size, although not perfectly so. In other words, if we know the size of a spreadsheet, we can use a reasonable CER to estimate how many errors the spreadsheet probably has.

### 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

Table 1 indicates that only two studies have looked at errors during cell entry (Lerch, 1988; Olson & Nilsen, 1987-1988), that is, when the developer is entering formulas. In these studies, errors were counted as they were made, so many of the errors counted would be corrected spontaneously by the developer. Still, they show that people do make many errors when they are working on spreadsheets.

### Errors at the End of the Development Stage

Most studies in Table 1 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), 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, Forthcoming) directly 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 almost identical. Even when a task was selected to be very simple and almost completely free of domain knowledge requirements (Panko & Sprague, Forthcoming; Teo & Tan, 1997), about 40% of all spreadsheets contained errors, and the CER was about 2%.

### Errors Found 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 (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.

Code inspection is very difficult. In programming experiments, it has been found that individual inspectors catch only half or fewer of all errors in program modules (Panko, 1997a). Even real-world group inspection usually only catches 80% or fewer of all errors in a program module (Panko, 1997a).

In code inspection, as in development, spreadsheeting

| Study | Sample | Study | Cell Error Rate (CER)* | Pct. of Models with Errors |
|---|---|---|---|---|
| **Field Audits** | | | | |
| Butler [1995] | 273 operational SSs | Only counted "material" errors. | | 10.7% |
| Cragg & King [1993] | 20 operational SSs from 10 firms. | 150 to 10,000 cells. Had been in use for median of 6 months | | 25% |
| Davies & Ikin [1987] | 19 operational SSs | 14 had qualitative errors. | | 21% |
| Hicks [1995] | 1 module with 19 submodules about to enter operation. | Part of a capital budgeting system for NYNEX. Code inspection by three analysts. Found 45 errors in 3,856 cells. | 1.2% | 26% of submodules |
| Cell Entry Experiments | | Errors counted when made, even if corrected later. | | |
| Olson & Nilsen [1987-1988] | 14 experienced Lotus 1-2-3 users. | 4 formula cells per person. 56 total. CER for formula cells. | 21% | |
| Floyd & Pyun [1987] | | Reanalyzed Olson & Nilsen [1985, 1987-1988] for errors in text cells. | 12.5% | |
| Lerch [1988] | 21 professionals. | CER based on formulas only. | 11.3% | |
| **Development Experiments** | | Errors counted at end of development process | | |
| Brown & Gould [1987] | 9 highly experienced SS developers | Developed 3 models apiece. All made an error in at least one model. | | 63% |
| Hassinen [1988] | 92 novice spreadsheet students. | Paper and pencil exercise. Formula cells only. | 4.3% | 55% |
| Hassinen [1988] | 10 novice students developing 48 SSs | Computer exercise for same task. | | 48% |
| Janvrin & Morrison [1996], Morrison [1995] | 61 upper division business & graduate accounting students | Study 1: CER based only on links between worksheets. Had 16 days to do task. Worked an average of 8.8 hours. | 7-14% | 84-95% |
| Janvrin & Morrison [1996] | 88 senior-level accounting students | Study 2: More complex multi-worksheet task. Again, only inter-worksheet links. | 8-17% | |
| Panko & Halverson [1997] | Undergraduate business students | Developed pro forma income statement based on the Galumpke task. Working alone, in groups of four. | 5.4%, 2.0% 2.0% | 80%, 60% |
| Panko & Sprague [1997] | 102 undergrad bus. students, 50 MBA students. 17 MBAs were experienced | Developed a model based on the Wall task designed to be relatively simple and free of domain knowledge. No difference in errors across groups. | 2.0% | 35% |
| Teo & Tan [1997] | 168 second-year IS students. | Developed a model based on the Wall task, then did a what-if analysis. | 2.0% | 42% |
| Panko (unpublished) | 80 undergraduate business students | Developed spreadsheet working alone or in triad (group of 3). CERs for individual, triad. | 4.6%, 1.0% | 86%, 27% |
| **Code Inspection Experiments** | Subjects looked for errors in a model. | CER is percent of errors NOT detected. | | |
| Galletta et al. [1993] | 30 MBA students and 30 CPA accountants | Examined 6 small SSs. Experienced developers did not find more errors. | 34-54% | |
| Galletta et al. [1996] | 113 MBA students | Finding eight seeded errors in a student budgeting model. | 45-55% | |
| Panko and Sprague [forthcoming] | 23 undergraduate subjects with errors in initial model. | Study described above. Code inspected own models. Fixed 18% of errors. Only 13% of spreadsheets were fixed completely. | 81% | |

*The Cell Error Rates (CER) is the percentage of cells containing errors.
See the Spreadsheet Research (SSR) Website for citations not listed in the References

**Table 1: Studies of Spreadsheet Errors**

has shown strong parallels to programming. In three published 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 et al., 1997; Panko & Sprague, Forthcoming).

We have a single example of real-world spreadsheet code inspection. In a personal communication with the author, Hicks (1995) described how a three-person team code inspect a large module (of 3,856 cells) from a larger spreadsheet. This effort found 45 errors, for a cell error rate of 1.2%. This is comparable to CERs seen in development experiments.

### 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.

Davies and Ikin (1987) inspected 19 operational spreadsheets. Four (21%) were found to have at least one serious quantitative error. 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.

Cragg and King 1993) inspected 20 operational spreadsheets. This audit found serious errors in 5 of the spreadsheets (25%). It also noted that six of the twenty had problems in the past. This study also found serious design flaws that could lead to errors later and rather weak development disciplines.

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. Individual inspectors found errors in 10.7% of the spreadsheets.

Although laboratory studies generally have higher error rates, field studies did not use methodologies that probably could find most errors. In the one field audit for which we have a cell error rate and used a methodology capable of catching a large fraction of all errors, (the study by Hicks) the CER (1.2%) is similar in magnitude to CERs seen in experiments.

### Does Groupwork Help?

In their ethnographic study of 11 spreadsheet developers, Nardi and Miller (1991) noted that all developers used at least one other person during development, either to provide technical or domain knowledge or to look the spreadsheet over for errors.

This led Panko and Halverson (1997) to examine group spreadsheet development. Their subjects developed a spreadsheet while 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. Even development with groups of four, furthermore, reduced errors no more than group spreadsheet code inspections are likely to do.

### Types of Errors

When many people think of spreadsheet errors, they think of simple mechanical errors, such as 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 mechanical errors are common, there are many other types of errors in spreadsheet development. Panko and 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. Teo and Tan (1997) demonstrated in an experiment how one type of qualitative error led to quantitative errors during what-if analysis.

Panko and Halverson (1996), following Allwood (1984), also found it useful to distinguish between three types of quantitative errors. In contrast to simple mechanical errors, described above, logic errors involve entering the wrong formula because of a mistake in reasoning. Logic errors also more difficult to detect and correct (Allwood, 1984) than mechanical errors. 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). Experiments (Panko & Halverson, 1997; Panko & Sprague, Forthcoming) have shown that all three forms of quantitative errors are common. 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.

## Development Practices and Policies

### Developer Surveys

While experiments and field audits provide useful insights into error rates, developer surveys provide complementary information by asking developers about the practices that they use when they develop spreadsheets.

The Davies and Ikin (1987) and Cragg and 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.

Earlier, we mentioned the ethnographic interviews of Nardi and Miller (1991). In general, they found that developers took extensive precautions when developing their spreadsheets. 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). While some had others look at their results, this is likely to be of only some use given the difficulties in finding errors even in full code inspection.

Later, Hendry and Green (1994) did a similar ethnographic study, this time with 10 spreadsheet developers. They also noted care in development, but when they asked subjects to select and explain one spreadsheet, they found that subjects often had a difficult time explaining parts of the spreadsheet.

Overall, both Nardi and Miller (Nardi & Miller, 1991) and Hendry and Green (1994) found that subjects took considerable pains when they built spreadsheets. However, these methods were largely informal. They fell far short of the disciplines long known to be necessary in programming, which has similar error rates.

Other studies have used questionnaire surveys to collect data from larger numbers of respondents. For instance, Schultheis and Sumner (1994) had 32 MBA students each discuss their most recent spreadsheet. They found that higher-risk spreadsheets used slightly more controls than lower-risk spreadsheets, but the use of controls was low in both cases.

On a larger scale, Floyd, Walls, and 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. Development was strongly iterative. There was a high level of confidence that the spreadsheet described was correct.

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. Respondents indicated that most of their spreadsheets were important, were run mostly on a regular basis, and produced results that were widely used by others. Spreadsheets were also complex. Yet developers used weak design methodologies. Testing was very limited, especially code inspection testing. Eighteen said that development had been rushed. For almost each control queried, 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 more than inexperienced developers, but the difference was only moderate.

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. Code inspection is very infrequent, and while execution 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 on spreadsheet development.

Galletta and Hufnagel (1992) surveyed 107 MIS executives using a mail questionnaire that focused broadly on policies for controlling end user development. The study found few restrictions on end user development and even fewer required post-development auditing. Where rules existed, furthermore, they tended to be widely ignored.

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." About 70% strongly agreed that this lack of testing standards was producing serious problems, and none disagreed.

Floyd, Walls, and Marr (1995) found that only about 15% of the firms had development policies and two-thirds development modification policies. None of the respondents reported a comprehensive set of policies for all phases of development. Furthermore, most policies were created and implemented at the workgroup level.

Speier and 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.

Finally, Hall ( 1996) asked respondents about corporate policies. Only 11% of the respondents knew of a comprehensive corporate policy for spreadsheet development. Of these, only one in three answered "Yes" when asked if they knew where to find the policy in written form! In 90% of the few cases where rules existed, furthermore, enforcement was left to the developer.

Overall, formal spreadsheet policies seem to be fairly rare, and their enforcement seems to be casual.

## 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. Brown and Gould (1987) and Davies and Ikin ( 1987), Panko and Halverson (1997), and Panko and Sprague (Forthcoming) all found high confidence despite numerous errors.

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 Reithel, Nichols, and Robinson (1996) found that their subjects rated large, well-formatted spreadsheets higher than they did small spreadsheets whether well formatted or not.

While such massive levels of overconfidence seem un-

reasonable, overconfidence is perhaps the most well-established phenomenon in the behavioral sciences, even among experts (Lichtenstein, Fischoff, & Philips, 1982; Plous, 1993). It is seen in almost every human cognitive activity and is even seen among many experts.

Overconfidence is corrosive because it tends to blind people to the need for taking steps to reduce risks. Overconfidence among spreadsheet developers may be the root cause of their failing to follow systematic development disciplines.

Overconfidence tends to be greatest when real-world subjects do not get detailed feedback over many cases (Shanteau & Phelps, 1977). Note, however, that mere experience is not enough. There must be very detailed feedback on each case. 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.

The experimental and field data suggest that people make many errors as they work. They catch some of these errors but certainly not all. They then fail to code inspect their spreadsheets, so they never see the errors they make. They are then confident because they see few errors after development and quickly ignore those errors as aberrations.

## Conclusions: 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.

We can also expect developers to make errors with other end user development tools. So for other tools as well, we need to conduct research on error categories and error rates. While many claims are made for the safety of new structured tools, many of these claims rest on such features as data typing, which would eliminate very few of the errors seen in spreadsheet experiments.

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. Several authors (Panko, 1997b) 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). One design method that may be useful is to have developers write out and critiqute at least difficult equations on paper before entering them into spreadsheet cells.

Most prescriptions for spreadsheet development have focused on the design and development phases, 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. As in programming, we will almost certainly have to impose unpleasant and expensive testing phases on spreadsheet development if we are really serious about reducing errors.

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 (1976) emphasized that code inspection should never be used for performance evaluation. As Beizer (1990) has emphasized, a climate of blaming will prevent program developers from acknowledging errors. Quite simply, although the error rates seen in research studies are appalling, we have seen that they are also in line with the normal accuracy limits of human information processing. They result from flaws in human cognition, and even the best and most careful people make too many errors for safe spreadsheeting. We cannot punish people for normal human failings.

Finally, we have been discussing developers in this paper, but 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.

### References

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

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

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

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

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.

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

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

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.

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*, 3(2), 79-95.

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

Hall, M. J. J. (1996, January). 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. (1995). NYNEX, personal communication via electronic mail.

Janvrin, D., & Morrison, J. (1996, January). Factors Influencing Risks and Outcomes in End-User Development. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, Kihei, Maui, Hawaii.

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., & 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 Software*, 11(2), 50-58.

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. (1997a). *A Human Error Data Website* (http://www.cba.hawaii.edu/panko/papers/humanerr/). Honolulu, HI: University of Hawaii.

Panko, R. R. (1997b). *Spreadsheet Research* (SSR) Website (http://www.cba.hawaii.edu/panko/ssr/). Honolulu, Hawaii: University of Hawaii.

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

Panko, R. R., & Sprague, R. H. J. (Forthcoming). Hitting the Wall: Errors in Developing and Code-Inspecting a "Simple" Spreadsheet Model. Accepted for Publication in *Decision Support Systems.*

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

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.

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 Judgment 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.

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

.

*Dr. Raymond R. Panko is a professor of information systems at the University of Hawaii. Before coming to the university, he was a project leader at SRI international, where he worked under Doug Engelbart, the inventor of the mouse and hyptertext. He has been conducting research on end user computing since the early 1970s, especially in the fields of electronic mail and videoconferencing. He first became interested in spreadsheet errors when he was preparing his text book, End User Computing: Management, Applications, and Technology, for Prentice-Hall.*