Routledge
Taylor & Francis Group

Journal of Management Information Systems

ISSN: 0742-1222 (Print) 1557-928X (Online) Journal homepage: https://www.tandfonline.com/loi/mmis20

Applying Code Inspection to Spreadsheet Testing

Raymond R. Panko

To cite this article: Raymond R. Panko (1999) Applying Code Inspection to Spreadsheet Testing, Journal of Management Information Systems, 16:2, 159-176, DOI: 10.1080/07421222.1999.11518250

To link to this article: https://doi.org/10.1080/07421222.1999.11518250

Applying Code Inspection to Spreadsheet Testing

RAYMOND R. PANKO

RAYMOND R. PANKO is Professor of Decision Sciences at the University of Hawaii. He has been conducting research on end user computing since the early 1970s, initially as a graduate student at Stanford University and later as a project manager at SRI International. His current focus is on the risks involved in computing, especially in spreadsheeting. He has created a website for research on spreadsheeting. Its URL is http://panko.cba.hawaii.edu/ssr/.

ABSTRACT: In programming, reliability requires an extensive testing phase. Spreadsheet development, which has about the error rate as program development, also needs to be followed by an extensive testing phase if spreadsheets are to be reliable. In this study, sixty undergraduate MIS students code-inspected a spreadsheet seeded with eight errors. They first inspected the spreadsheet working alone. They then met in twenty groups of three to reinspect the spreadsheet together. Effort was made to prevent hasty inspection.

Individual code inspection, consistent with past studies of both spreadsheet and program code inspection, caught only 63 percent of the errors. Group inspection raised this to 83 percent. However, the group phase never found new errors; it merely pooled the errors found during the individual phase by the three members. One group even "lost" an error found during the individual phase. This raises the question of whether a group code inspection phase is really necessary. Other findings were that subjects were overconfident when inspecting alone, that certain types of errors are especially difficult to detect, and that the benefits of the group phase is greatest for these difficult-to-detect types of errors.

KEY WORDS AND PHRASES: code inspection, error detection, spreadsheet errors, spreadsheet testing.

SPREADSHEETING IS ENORMOUSLY IMPORTANT IN ORGANIZATIONS. Based on sales figures, tens of millions of managers and professionals around the world create hundreds of millions of spreadsheets each year. At least some of these spreadsheets are very large [14, 18] and very complex [18]. Most important, many mission-critical corporate decisions are guided by the results of large and complex spreadsheets.

Spreadsheet Errors and Errors in Other Human Cognitive Activities

If organizations are to depend on the results of spreadsheets, then these spreadsheets must be very accurate. Such accuracy, unfortunately, is very difficult to achieve.

Journal of Management Information Systems / Fall 1999, Vol. 16, No. 2, pp. 159–176.

0742–1222 / 1999 $9.50 + 0.00.

In programming, errors (faults) in a program may be in sections of code that are never or only rarely executed in practice. Consequently, the failure rate may be far lower than the fault rate. In spreadsheeting, however, almost every numerical and formula cell is on a computational cascade leading to a bottom-line value. As a result, almost any cell error in a spreadsheet will lead to an incorrect bottom-line value.

Accuracy in large spreadsheets requires an extremely low error rate. For instance, if a large spreadsheet with a thousand or more cells is to be error-free, then the average error rate in development must be better than one error in every thousand cells. If *most* such large spreadsheets are to be error-free, in turn, the development error rate in general must be one-tenth to one-hundredreth lower.

Such low human error rates, however, would be unprecedented. Data on human cognitive errors [31] from a broad spectrum of activity domains ranging from typing to mathematical problem solving and programming indicate that it would be more reasonable to expect something on the order of one error in every *hundred* cells. In programming, for instance, we have data from thousands of real-world programs showing that a typical frequency of errors (faults) is around five in every hundred lines of noncomment source [32].

Indeed, in recent years, several studies have specifically examined errors in spreadsheet development and auditing [31]. These studies include over a dozen laboratory experiments on over a thousand subjects. They also include four field audits of more than 300 real-world spreadsheets. Every study, without exception, has found error rates much higher than organizations wish to tolerate. In particular, studies that have measured the *cell error rate* (CER)—the percentage of numerical and formula cells containing errors—have always found CERs in excess of 1 percent. As just noted, this is about what one would expect from other cognitive research on human error rates [30].

The convergence of error rates across cognitive activities is not surprising. An emerging theoretical framework explains both correct performance and errors across a broad spectrum of cognitive tasks [2, 43]. It explains at least the main types of errors that people make [2, 43].

In essence, the framework shows that, although human beings can think very quickly [43] and can juggle multiple tasks [19], we are only able to achieve human performance levels because our brains take a number of "shortcuts" that inevitably lead to occasional errors [43]. In other words, a small error rate is an inevitable aspect of human cognition, not merely a result of carelessness, rushing, or other blameful actions. Although we do not err frequently, we all err a small percentage of the times we take cognitive actions.

Unfortunately, the emerging framework does not predict specific human error rates. Although empirical studies are highly convergent in the error rates they have found [30], error rates must be measured in each new cognitive domain being studied. In spreadsheeting, the focus to date has been on the measurement of *development* errors. It is now time to focus more specifically on expanding the limited research that has been done on errors in spreadsheet *testing*.

Unfortunately, existing error frameworks focus primarily on human error *production*.

In error detection, which is more directly relevant to testing, the theoretical situation is much weaker. Reason [43] puts it this way: "Despite the obvious importance of the topic, the psychological literature contains very little in the way of empirical studies of error detection or of theories to explain the process by which people catch and correct errors made by themselves or others" (p. 148).

Code Inspection

When a programmer finishes a program module, he or she knows that it almost certainly contains errors (faults) and indeed probably contains errors in about 5 percent of all lines of code [30]. This is far too high. As a result, development is followed by an aggressive testing phase that may take up to a third of total development time and effort.

One method of program testing is execution testing. Known data are run through the module. The tester then compares the program's output with known results. In spreadsheeting, unfortunately, it is common for the spreadsheet to model more complex situations than have ever been modeled before. As a result, it is uncommon to have the known data and results required for execution testing.

Another popular method of program testing is *code inspection*, in which the program's code is examined in detail to find errors. Code inspection in programming can catch up to about 80 percent of all errors in real-world programs [30]. Code inspection is a promising area to explore because it is rare in operational end-user spreadsheet creation [8, 18]. If we can demonstrate that code inspection is effective in spreadsheet testing, this might be able to motivate the expansion of code inspection in real-world spreadsheet creation.

Inspection in programming has a long history, dating back to the early work of Fagan [12]. There generally are two types of inspection: inspection of requirements and design documents and inspection of actual code. There may also be two types of code inspection—one at the module level and one at the aggregated-system level. In other words, inspection may occur multiple times over the systems development life cycle.

One tenet of most code inspection methodologies is the collection and publication of data on error rates. Data from many programming code inspections have been made public [30], including studies that have varied procedures to determine the impacts of different procedures on yield rates [30]. More recently, there have been a number of experiments that have systematically varied procedures under highly controlled conditions [4, 23, 28, 35, 37, 38, 39].

The research has found consistently that code inspection can catch many, but not all, of the errors in a document or program. As noted above, for real-world code inspection, group detection rates of around 80 percent have been claimed [30]. Some laboratory studies have had lower group detection rates [23, 36, 37]. Given the error rates seen in spreadsheet development studies [31], however, even the lowest gains seen in experiments would be welcome.

Although code inspection was developed in programming, the need for a systematic error checking phase has been documented in other areas of human cognition. When

Allwood [1] studied statistical problem solving, he noted that subjects often stopped working because they had a definite or vague suspicion that an error had occurred. At other times, however, they stopped to check their work even when they did not suspect an error. Allwood called this nonprompted error search a "standard check." He found that subjects caught almost no high-level knowledge-based errors outside of standard check episodes. In observational studies of writing, furthermore, Hayes and Flower [19] noted that writers often stopped to "review" what they had written, checking systematically for errors. These reviews often occurred during the writing process itself, but there were also systematic reviews after drafting work.

In programming, it is likely that programmers often engage in systematic standard checks during the actual development of the program, although close studies in programming have not focused on this issue. Code inspection, then, is like reviewing a draft document. In programming, almost all methodologies require that inspection be done by groups rather than by individuals. This is because individual error detection rates are too low. The discoveries of several individuals need to be pooled to yield acceptable yield (error detection) rates. Group code inspection methodologies usually specify two phases. In the first phase, the individuals in the team study the module working alone. In the next phase, they meet as a group.

Different methodologies specify different tasks in the two phases. Fagan [12], for instance, suggested that individuals should merely familiarize themselves with the module in the first phase. Only in the face-to-face team meeting would error discovery be the key goal. Other methodologies have inspectors focus on error detection in the first phase, so that the focus of the team meeting is on the compilation of errors, although new errors are also sought during these face-to-face meeting phases [37, 38].

In spreadsheeting, only one field audit has used something like a standard two-phase team code inspection methodology. Hicks [21] described a three-person code inspection of a 3,850-cell module of a larger capital budgeting spreadsheet that was about to become operational at NYNEX. Although this spreadsheet was very large compared with recommended maximum sizes for code inspection in programming, the spreadsheet had a good deal of redundancy because of formula copying across columns. This inspection found errors in 1.2 percent of the cells, which is similar to the cell error rate in laboratory spreadsheet development experiments [31] and in programming [30]

Also in spreadsheeting, Galletta and his colleagues [15, 16, 17] have conducted two laboratory experiments in spreadsheet *auditing*, which is similar to code inspection except in one important way. There is only an individual phase.

In these two experiments, subjects caught only about half of all seeded errors in the spreadsheets they inspected. The first study [15] specifically examined whether spreadsheet development experience improved error finding. The data showed that experienced spreadsheet developers finished more quickly than inexperienced subjects but did not find more errors than novices.

Panko and Sprague [34] also found low error detection rates in a study using a similar methodology. They also found no significant differences in spreadsheet *development* error rates across undergraduates, M.B.A. students with little spreadsheet develop-

ment experiments, and M.B.A. students with substantial spreadsheet development experience.

Key Variables

Galletta and his colleagues [15, 16, 17] emphasized both accuracy and *speed* in their instructions. Subjects were told to be accurate but also to work quickly. Speed has the obvious benefit of reducing cost and must be considered.

However, inspection studies in programming consistently find that error detection rates fall as inspection speed increases.

- Basili and Perricone [3] examined FORTRAN programs in a software engineering laboratory. When the inspection rate was limited to 50 lines of code per hour, the inspectors found errors in 1.6 percent of all statements. When the inspection rate tripled to 150 lines per hour, however, they only found errors in 1.2 percent. When the inspection rate passed 200 lines per hour, furthermore, they only found errors in 0.6 percent of all lines of code.
- Russell [44] looked at inspection rates of 150, 450, and 750 lines of code per hour. The respective error rates discovered were 3.7 percent, 1.5 percent, and 0.8 percent.
- Ebenau and Strauss [11] found that the error rates in "nonhasty" and "hasty" code inspections were 2.0 percent and 1.3 percent, respectively.
- Weller [48] presented data on three-person code inspection teams. He found that, at less than 200 lines per hour, the detected error rate was 2.4 percent. Above 200 lines of code per hour, it fell to 2.0 percent. Four-member teams did somewhat better, but their detected error rate still fell from 3.1 percent to 2.5 percent when the inspection speed passed 200 lines of code per hour.

More generally, speed-accuracy tradeoffs have been seen in other human cognitive domains [26].

Since there may be a similar pattern in spreadsheet inspection, it may be good to emphasize only accuracy in some studies until the issue can be resolved empirically. One approach to doing this is to impose a *minimum* inspection time so that subjects will not be able to rush through the inspection.

There are two basic types of errors [32]: *Quantitative* errors produce incorrect bottom line values; *qualitative* errors, in turn, are flaws in the spreadsheet design that may lead to errors in the future but do not corrupt the current bottom-line values. Our study focuses only on quantitative errors. When we use the term "error" by itself, we mean quantitative errors.

People make various types of quantitative errors when they work. Several error researchers have produced useful taxonomies of error types [1, 29, 41]. Allwood [1] divided errors into mechanical, logical, and omission errors. Panko and Halverson [32] use those terms but define how they apply to spreadsheet development. (In particular, their category "omission errors" is somewhat different from Allwood's category of the same name.)

- *Mechanical errors* are simple slips, such as mistyping a number or pointing to the wrong cell when entering a formula.
- *Logic errors* are "thinking errors." They include having the wrong algorithm for creating the formula or having the right algorithm but creating a formula that does not correctly implement the process.
- *Omission errors* occur when the developer leaves something out of the spreadsheet that is in the problem statement. Omission errors seem especially resistant to discovery [1], so we would expect the detection rate for omission errors to be lower than those of mechanical or logic errors.

We need to consider error type in inspection research because Allwood [1] and others have shown that commission and detection rates vary by error type. Consequently, if we examine error rate by error type, we should get a better picture of error detection in inspection than we would if we simply measured the gross error detection rate. We need to know specifically which types of errors are caught well in code inspection and which types are relatively immune to inspection detection.

It also seems prudent to consider the length of cell formulas. In a study of proofreading, Healey [20] compared error detection rate with word length. His subjects searched a document for *nonword* spelling errors, that is, for misspellings that were not valid words themselves. Healey found that, for short words of two to four letters, the error detection rate was very high, ranging from 92 percent to 99 percent. However, for words of five letters, it fell to 77 percent, and for longer words the detection rate was only 72 percent. If people find it difficult to detect simple nonword spelling errors in longer words, then it seems plausible that people will find the detection of errors in longer formulas difficult.

Another concern in spreadsheet code inspection research is overconfidence. In past studies of spreadsheeting, subjects have been highly confident in their results, even when their error rates were quite high [6, 8, 9, 14, 34]. Although disturbing, this is not surprising, because humans tend to be overconfident across a wide variety of situations [7, 25, 35, 45, 47]. Overconfidence is a concern because we tend to act on our metacognition (beliefs about what we know) [27]. In particular, our stopping rules for when we will cease searching for errors [42] are likely to be shorter if we are overconfident.

Overconfidence, by the way, is not just a problem of novices. Numerous studies [22, 45] have shown that even experts are often badly overconfident. However, overconfidence is not always the case for experts. Shanteau [45] found that a common denominator among professions that were not overconfident is that they both received systematic feedback and constantly analyzed feedback data for individual cases. Other studies [24, 35] documented the importance of feedback information in laboratory settings. Code inspection, by confronting the programmer or spreadsheet developer with specific frequencies of error, should be able to provide that kind of needed feedback. Unfortunately, as noted above, spreadsheet developers rarely conduct formal code inspections, so they do not receive that type of feedback.

Research Goals

Individual and Group Error Detection

OUR MAIN RESEARCH GOAL, OF COURSE, WAS TO DETERMINE the percentage of errors that subjects would detect. We wished to measure the percentage of errors found by individuals. We also wished to measure the extent to which groups found more errors than individuals. This led to hypothesis 1:

> *H1: Groups will detect more errors per spreadsheet than will individuals.*

To test this hypothesis, we compared the number of errors per spreadsheet discovered by individuals and groups using a one-tailed *t*-test. We used the traditional 0.05 confidence level for this and other hypothesis tests.

Types of Errors

We also wished to determine the percentage of subjects detecting different *types* of errors. This led to hypotheses 2 and 3:

> *H2: Mechanical error detection rates for individuals will be higher than omission error detection rates for individuals.*

To test hypothesis 2, we counted each error for each individual as a 0 if it was not discovered or as a 1 if it was discovered. We then compared the total distribution for omission error detections across subjects (120 cells) with the total distribution for mechanical errors (240 cells), using a one-tailed *t*-test.

> *H3: Mechanical error detection rates for groups will be higher than omission error detection rates for individuals.*

We tested this hypothesis in the same way we tested hypothesis 2, using group data.

Formula Length

As noted earlier, we were concerned with the length of the formulas, in terms of arithmetic operations and cell references because of reduced error detection rates when people proofread long words. This led to hypotheses 4 and 5:

> *H4: Mechanical errors will be detected more frequently in short formulas (with four or fewer elements) than in long formulas for individuals.*
>
> *H5: Mechanical errors will be detected more frequently in short formulas (with four or fewer elements) than in long formulas for groups.*

These hypotheses were tested in the same way that hypotheses 2 and 3 were tested.

Overconfidence

We were also concerned with overconfidence, given the broad range of studies that have measured overconfidence and the fact that overconfidence can cause us to limit our search for errors. This led to hypotheses 6 and 7.

> *H6: For subjects working alone, the estimated percentage of errors found will be greater than the actual percentage of errors found.*
>
> *H7: For subjects working in the group phase, the estimated percentage of errors found will be greater than the actual percentage of errors found.*

To test this hypothesis, we compared the estimated percentage of errors with the actual percentage of errors using a one-tailed *paired t*-test.

Method

The Code to Be Inspected

THE SPREADSHEET THAT SUBJECTS INSPECTED WAS ADAPTED from one developed by Galletta et al. [16, 17]. Figure 1 shows the spreadsheet used in our study. As in one treatment in Galletta et al. [16, 17], the printed form had both the result and the formula in each formula cell.

For our experiment, we modified their spreadsheet in three ways. First, we added two omission errors (the original spreadsheet had no omission errors). Second, we made it look like an Excel spreadsheet instead of the Lotus spreadsheet it was originally. Third, we dropped two errors, namely, the treatment of starting and ending cash in the totals column. A pretest of sixteen subjects found that students who made the two errors believed that they were correct even after they were told the correct procedure. In the end, we had a spreadsheet seeded with two omission errors, with four mechanical errors in formulas with three operations and cell references, and two more mechanical errors in formulas with six and nine operations and cell references. The model had no logic errors.

Sample

The study used thirty third- and fourth-year undergraduate MIS majors taking networking classes. Subjects received extra credit for participation. All had previously taken a class that taught spreadsheet skills. None had trouble understanding the spreadsheet's formulas.

The use of student subjects is always a concern. However, as noted earlier, two previous studies have suggested that students should be fairly good surrogates of experienced spreadsheet developers. First, in their code inspection study, Galletta et al. [15] found that inexperienced and experienced spreadsheet developers had very similar error rates. Second, Panko and Sprague [34] compared spreadsheet development for undergraduate MIS majors, M.B.A.s with little spreadsheet development

	A	B	C	D	E	F	G
1	Cash Budget			2%	Inflation/ semester		
2				Fall	Spring	Summer	Overall
3	Cash, beginning			$1,000	$1,360 =D8	$673 =E8	
4	Outflows - School costs			$4,468 =sum(D10..D12)	$4,474 =sum(E10..E12)	$4,480 =sum(F10..F12)	$13,422 =sum(D4..F4)
5	Living costs			$4,172 =sum(D14..D20)	$4,213 =sum(E14..E20)	$4,485 =sum(F12..F19)	$12,870 =sum(D5..F5)
6	Inflows - Loans			$3,000	$3,000	$3,000	$9,000 =sum(D6..F6)
7	Support from home			$6,000 =ROUND(1000+ D4+D5-D3-D6-3)	$5,000 =ROUND(1000+ E4+E5+E3-E6-3)	$6,000 =ROUND(1000+ F4+F5-F3-F6-3)	$17,000 =SUM(D7..F7)
8	Cash, end			$1,360 =D3-D4- D5+D6+D7	$673 =E3-E4-E5+E6+E7	$980 =F3-F4-E5+F6+F7	$3,013 =sum(D8..F8)
9				=======	=======	=======	=======
10	School (contractual): Tuition			$4,115	$4,115	$4,115	$12,345 =SUM(D10..F10)
11	Fees			$53	$53	$53	$159 =SUM(D11..F11)
12	School (other): books/supplies			$300	$306 =ROUND (D12*(1+$D$1),0)	$312 =ROUND (E12*(1+$D$1),0)	$918 =SUM(D12..F12)
13	Living (contractual)	MONTHLY	MONTHS				
14	Housing	$450	4	$1,800 =C14*B14	$1,800 =D14	$1,800 =D14	$5,400 =SUM(D14..F14)
15	Insurance	$53	4	$212 =C15*B15	$212 =D15	$212 =D15	$636 =SUM(D15..F15)
16	Living Costs (other)						
17	Food	$330 =11*30	4	$1,320 =C17*B17	$1,346 =ROUND (D17*(1+$D$1),0)	$1,373 =ROUND (E17*(1+$D$1),0)	$4,039 =SUM(D17..F17)
18	Entertainment	$150	4	$600 =C18*B18	$612 =ROUND (D18*(1+$D$1),0)	$624 =ROUND (E18*(1+$D$1),0)	$1,836 =SUM(D18..F18)
19	Transportation	$40	4	$160 =C19*B19	$161 =ROUND (D19+(1+$D$1),0)	$164 =ROUND (E19*(1+$D$1),0)	$485 =SUM(D19..F19)
20	Clothing	$21	4	$80 =C20*20	$82 =ROUND (D20*(1+$D$1),0)	$84 =ROUND (E20*(1+$D$1),0)	$250 =SUM(C20..F20)

Figure 1. The Spreadsheet with Seeded Errors

experience, and spreadsheet developers with extensive development experience. They found that differences between groups were neither statistically significant nor of practical importance. Finally, our detection rate during group code inspection (83 percent) was very similar to the 80 percent yield rate found in program code inspection by experienced programmers inspecting operation programs in industry [30].

Assignment to Groups

Subjects were scheduled to appear in one of three time periods. Subjects could select the time period. Subjects were assigned to groups randomly after arrival.

Process

Initial Steps

After the subjects arrived, the author explained the purpose of the experiment, namely, to see if subjects would catch all of the seeded errors and, if not, how many they would catch. Subjects were then given a questionnaire about themselves and their spreadsheeting experience. They were also given five minutes of instruction in code inspection, including the need to relate the problem statement to the spreadsheet and the need to examine every cell and every arithmetic operation and cell reference in every cell. They were also told that, if they rushed, they would miss many errors and so they must work slowly. They were told that there would be two phases: an individual phase and a group phase. After a question period, they filled out agreements to participate.

Individual Code Inspection Phase

In the first (individual) phase, subjects were given a booklet with the task statement and sheets on which to record the errors they found and their correction for each error. A correction was requested to ensure that they understood the nature of the error.

Subjects were told that they would have to take a full forty-five minutes to do the code inspection and that if they finished early they should go back and look for more errors. As noted earlier, hasty code inspection in programming has long been associated with low error detection rates. A pretest suggested that forty-five minutes represents a slow and deliberate rate of inspection of the spreadsheet used in the experiment.

Subjects noted their starting times. They also recorded their stopping times to ensure that they had met the minimum time requirement. They then estimated, using a multiple-choice scale consisting of 5 percent, 10 percent, 20 percent, 30 percent, 40 percent, 50 percent, 60 percent, 70 percent, 80 percent, 90 percent, 95 percent, and 100 percent, what percentage of the errors in the spreadsheet they had discovered.

Group Code Inspection Phase

At the end of the individual code inspection phase, subjects had a ten-minute break. They were then given the group code inspection task. They were told, consistent with programming code inspection practice, that one person would be the recorder of errors and another person would be the reader, who would go through the program cell by cell, announcing its purpose and asking members to check it for errors. They were told that they could refer to their list of errors found during the individual inspection phase but they should not limit themselves to this. They should again go through the model cell by cell, this time as a group. The groups worked in separate breakout rooms.

The experimenter circulated constantly from room to room, enforcing the requirement to inspect the model cell by cell and reminding subjects to use the full forty-five

minutes allotted to the task. In practice, this proved impossible to enforce. All groups essentially stopped at thirty-five to forty minutes of real work, doing desultory checking afterward. They felt that they had gone through the model slowly and carefully and that additional time was a waste of time. Observation of the groups suggested that they had indeed been working slowly and deliberately during the group phase.

At the end of the period, subjects were told to estimate independently the percentage of errors they thought *their group* had found. To preserve independence, group members were not allowed to see one another's estimates.

Analysis of Errors

To analyze the errors, the author examined the error detections/corrections against a list of eight seeded errors. Each individual and each group was scored as detecting or not detecting each error.

We use parametric statistics to report the results. We also did the analysis with nonparametric statistics. However, the probabilities were almost identical, and most readers are more familiar with the parametric *t*-tests reported in this paper.

Results

TABLE 1 SHOWS THE RESULTS OF THE HYPOTHESIS TESTS. This section looks at the numbers behind this table.

Comparison with Past Results

Table 2 shows that subjects working alone detected 63 percent of all seeded errors. This is a higher percentage than Galletta et al. [16, 17] found in their similar task. This difference may be due to modifications to the task. The difference, however, between our yield rates and those of Galletta et al. [16, 17] was modest. Moreover, our detection rates were fairly close to the error detection rates found in several previous studies of individual program code inspection with experienced programmers and students [4, 28]. At the same time, our detection rates were higher than those seen in other programming inspection studies [37].

Table 2 gives additional detail on error detection rates for individual questions. It shows the percentage of individuals detecting each error, the percentage of groups detecting each error, and the percentage gain in group detection compared with individual detection for each error. The table also shows the total percentage of errors that subjects estimated they detected and the degree to which estimates exceeded actual yields.

Group Versus Individual Code Inspection Overall

Group code inspection overall, as expected, did better than individual code inspection. Groups of three found 83 percent of all errors, while individuals found only 63 percent.

Table 1. Results of Hypothesis Tests

H1	Groups will detect more errors per spreadsheet than will individuals.	0.009
H2	Mechanical error detection rates for individuals will be higher than omission error detection rates for individuals.	< 0.001
H3	Mechanical error detection rates for groups will be higher than omission error detection rates for groups.	< 0.001
H4	Mechanical errors will be detected more frequently in short formulas (with four or fewer elements) than in long formulas for individuals.	< 0.001
H5	Mechanical errors will be detected more frequently in short formulas (with four or fewer elements) than in long formulas for groups.	< 0.001
H6	For subjects working alone, the estimated percentage of errors found will be greater than the actual percentage of errors found.	0.004
H7	For subjects working in the group phase, the estimated percentage of errors found will be greater than the actual percentage of errors found.	0.056 (failed)

Number of individuals: 60; number of groups of three: 20.

This was significantly better than individual detection at the 0.009 level, so hypothesis 1 was supported.

Six of the twenty groups caught *all* errors, and half found at least seven of the eight errors. This indicates a significant ceiling effect, which limits statistical significance testing. At the same time, to be successful, real-world group inspection should catch a large majority of all errors, so to use a task without a ceiling effect would also be problematic. In addition, a ceiling effect reduces significance levels, so its presence actually strengthens our confidence in the significant results shown in Table 1.

Groups recorded 31 percent more errors than individuals, but this gain was not uniform across all types of error. For errors in longer formulas, the gains were 50 percent and 54 percent. For omission errors, the gains were 57 percent and 83 percent. *In other words, the group phase is most successful precisely for the errors that individuals detect the least frequently.* Although this result is dictated by probability, it is something to keep in mind when considering the value of group inspection.

On closer examination, however, groups did not report *any* errors not previously reported by their group members during the individual phase. In other words, the face-to-face code inspection did not find anything more than the subjects would have found had they merely collected the errors found by individuals during the first phase. In fact, one group even failed to report an error found by one of its members during the individual phase. Therefore, the gain from "groupwork" came simply from pooling the different errors detected previously by the three members, and was not even perfectly successful at that.

This lack of gain during the group meeting phase raises the issue of whether a

Table 2. Error Detection Rates by Type of Error

Cell	Description of error	Yield alone (%)	Yield in groups (%)	Gain for groups (%)
None (a)	Parental gift omitted from the spreadsheet	30	55	83
None (a)	Parking expense omitted from the spreadsheet	35	55	57
E19 (b)	=ROUND(D19+(1+$D$1),0). D19+ should be D19*.	47	70	50
F8 (b)	=F3=F4-F5+E6+E7. Should be =F3=F4-E5+E6+E7.	58	90	54
B17	=11*30. Should be =12*30.	88	100	13
D20	=C20*20. Should be =C20*B20.	85	100	18
F5	=SUM(F12..F19). Should be =SUM(F14..F20).	82	100	22
G20	=SUM(C20..F20). Should be =SUM(C19..F20).	82	95	16
	Overall	63	83	31
Estimated percentage of errors found		73	86	
Average overconfidence		16	4	

(a) Omission error; (b) long formula mechanical error.
Number of subjects working alone (first phase): 60; number of groups of three (second phase): 20.

face-to-face meeting is really needed. This issue has been discussed considerably in the programming literature [38, 46]. At the same time, however, someone must collect the errors recorded during the individual code inspection phase. The groups at least recorded all but one error. Would other ways of collecting errors be as good?

Omission Versus Mechanical Errors

As expected, subjects had a difficult time detecting the two omission errors. When working alone, only 30 percent and 35 percent of the subjects detected the two errors. This was far below their mechanical error detection rates, which ranged from 47 percent to 85 percent. Hypothesis 2 was supported at beyond the 0.001 probability level.

During the group phase, the two omission errors were also detected less frequently than were mechanical errors, both being detected by only 55 percent of the groups. Hypothesis 3 was supported at beyond the 0.001 probability level.

Longer and Shorter Formula Detection Rates for Mechanical Errors

For individuals working alone, lower error detection rates were also seen for the two mechanical formula errors that had more than three operations and cell references.

One was detected by only 47 percent of the subjects, the other by only 58 percent. In contrast, the four formulas with only three operations and cell references were detected by 82 percent to 88 percent of all individuals. The difference in detection rates between errors with three elements and those with six or more elements was significant, with a probability below 0.0001. Hypothesis 4 was supported.

In the group phase, all the groups detected three of the four short formulas, and 95 percent found the remaining one. The average detection rate for all four mechanical errors with short formulas was 99 percent. For the two longer formulas with mechanical errors, however, the group detection rates were only 70 percent and 90 percent. Hypothesis 5 was supported at beyond the 0.001 confidence level.

Overconfidence

As expected, subjects were overconfident when working alone. While subjects found only 63 percent of the errors, they estimated that they would find 73 percent. The difference between estimates and reality was significant at the 0.004 level, so hypothesis 6 was confirmed. More specifically, 61 percent overestimated the percentage of errors they found, while only 27 percent were underconfident.

For groups, subjects were only overconfident by four percentage points. Therefore, hypothesis 7 failed, although its probability was close to significant at 0.056.

In overconfidence studies, one of the most widely reported patterns is the hard-easy effect [5, 10, 25, 40]. When performance is high, confidence is also high, and overconfidence is low or nonexistent. However, as task difficulty increases, so that performance drops, confidence usually either stays constant or decreases much less than performance. In other words, overconfidence increases as performance decreases. Our subjects' answers definitely displayed this pattern. With some exceptions, subjects who found most or all errors had high confidence and were not overconfident. However, confidence generally remained high for subjects who found fewer errors. Overall, then, subjective estimates of performance in spreadsheet code inspection should not be taken as indicators of likely performance.

Conclusion

THIS STUDY CONTINUES THE PATTERN SEEN IN PAST STUDIES of spreadsheet error—namely, that spreadsheet errors occur at about the same rate as programming errors. Most past studies looked at errors during development or during single-phase (individual) code inspection. This study extends the research to two-phase, individual-group code inspection, in which members of a team first inspect the spreadsheet individually and then meet as a group to inspect it again.

Did teamwork help? The short answer is that it did. During the group phase, teams of three found 83 percent of all errors, while individuals working alone only found 63 percent percent. This is a 31 percent improvement.

More important, the group phase was particularly effective at finding errors that individuals found difficult to detect, including omission errors and mechanical errors

in longer formulas. For such detection-resistant errors, we found improvements of 50 percent to 83 percent for group inspection compared with individuals working alone. In general, more attention should be given to detection rates for different types of errors in all code inspection studies. The fact that the group phase does little better than individual inspection for easily detected errors tends to overshadow a potentially critical role in finding difficult errors if one only looks at gross error rates.

Although the gain from individual detection rates to group detection rates was impressive, it was only as good as we could have done by pooling the errors found by its members during the individual phase. The groups did not discover any new errors during their face-to-face meeting, despite cell-by-cell reinspection of the spreadsheet. In one case, a group even failed to record an error found by one of its members during the individual phase. This pattern of little or no gain and actual loss in the group phase has also been seen in some studies of programming code inspection [36]. Our findings suggest, consistent with past programming research, that a group phase may not be necessary. We may be able to do something like nominal group analysis, in which we merely compile a combined list of errors found by individuals. Tjahjono [23] found that tools such as nominal group analysis do, in fact, appear to work well.

The fact remains, however, that *someone* has to select the final list of errors. In experiments with seeded errors, the experimenter can do this easily. In the real world, things are not likely to be as simple. A simple pooled list of individual errors would contain numerous false positives—identified "errors" that are not really errors. Fixing these false positives would waste time and, since they would in some cases actually be wrong, could introduce new errors. The value of the group phase is worth pursuing.

Another broad consideration is that even team code inspection failed to catch 17 percent of the simple, easily demonstrable errors seeded into our spreadsheet model. If we had included Cassandra logic errors [32], which it is difficult to prove are errors even if identified, face-to-face groups might have rejected such errors even if they were presented by a group member. This group rejection of Cassandra errors has been seen before in development research [33].

Finally, although a 17 percent miss rate for groups is better than the 37 percent miss rate for individuals, it still leaves many errors in larger spreadsheets. Even the fact that a ceiling effect limited the gain from individual to group work does not make code inspection look like the mythic silver bullet that will remove all errors. Code inspection looks promising, but it does not look like a cure for spreadsheet error, any more than it has proven a cure for errors in programming.

Moreover, as just noted, we found that group code inspection yield rates are even worse for certain types of errors. In this study, we specifically looked at omission errors and at mechanical errors in longer formulas. Consistent with research done in other areas, the group's success in finding these two types of errors was only 55 percent and 80 percent, respectively. Furthermore, the study did not look at logic errors, some of which, we have just seen, groups find very difficult to agree upon even if they are detected by one member [33]. A lower detection rate for certain types of errors suggests that code inspection is not as good as its overall success rate indicates.

This study looked at several variables that we hope will be considered in future code

inspection research. First, our study examined overconfidence. If people are overconfident, they may stop their error detection work too soon, both overall and while inspecting individual cells (or lines of code). Our subjects were indeed moderately overconfident, as expected from past research on overconfidence. Although the overconfidence was only moderate, we should see if reducing overconfidence by giving detailed feedback will improve error detection.

Overall, group code inspection did substantially better than individual code inspection, suggesting that it should be used in the field. However, implementing a code inspection discipline is likely to be very difficult. Spreadsheet development usually is not performed by information systems professionals. Rather, it is done by ordinary managers and professionals in functional areas, for whom spreadsheet development is a small part of their overall jobs. Potential testers with sufficient functional knowledge to aid in the inspection would also have to be functional professionals who would have to take time away from their "main work." These organizational considerations, along with a strong sense of denial of error on the part of spreadsheet developers, probably explain why inspection is so rare in practice today and why it is likely to be difficult to introduce in the future. In effect, spreadsheet developers are roughly where programmers were in the 1950s and 1960s. Somehow, we must teach new dogs (spreadsheet developers) old tricks.

A final point may also be the most important. As noted at the beginning of the paper, many spreadsheets are very large, containing thousands of cells. For nearly complete accuracy, we would need to greatly reduce observed cell error rates (1 percent to 5 percent), which are similar to fault rates in programming, to perhaps one error in ten thousand cells. Our code inspection results, which are consistent with group code inspection studies in programming, suggest that an 80 percent reduction is about all we are likely to achieve. Quite simply, even with code inspection, spreadsheets *will* remain unsafe in the future.

REFERENCES

1. Allwood, C.M. Error detection processes in statistical problem solving. *Cognitive Science, 8*, 4 (1984), 413–437.

2. Baars, B.J. A new ideomotor theory of voluntary control. In B.J. Baars (ed.), *Experimental Slips and Human Error.* New York: Plenum, 1992, pp. 93–120.

3. Basili, V.R., and Perricone, B.T. Software errors and complexity: an empirical investigation. In M. Sheppard (ed.), *Software Engineering Metrics, Vol. I: Measures and Validation.* Berkshire, UK: McGraw-Hill, 1993, pp. 168–183.

4. Basili, V.R., and Selby, R.W., Jr. Four applications of a software data collection and analysis methodology. In J.K. Skwirzynski (ed.), *Software System Design Methodology.* Berlin: Springer-Verlag, 1986, pp. 3–33.

5. Bolger, F., and Wright, G. Reliability and validity in expert judgment. In G. Wright and F. Bolger (eds.), *Expertise and Decision Support.* New York: Plenum, 1992, pp. 47–76.

6. Brown, P.S., and Gould, J.D. An experimental study of people creating spreadsheets. *ACM Transactions on Office Information Systems, 5*, 3 (1987), 258–272.

7. Camerer, C.F., and Johnson, E.C. The process-performance paradox in expert judgment: how can experts know so much and predict so badly? In K.A. Ericsson and J. Smith (eds.), *Toward a General Theory of Expertise.* Cambridge: Cambridge University Press, 1991, pp. 195–217.

8. Cragg, P.G., and King, M. Spreadsheet modelling abuse: an opportunity for OR? *Journal of the Operational Research Society, 44*, 8 (1993), 743–752.

9. Davies, N., and Ikin, C. Auditing spreadsheets. *Australian Account* (1987), 54–56.

10. Dunning, D.; Griffin, D.W.; Milojkovic, J.D.; and Ross, L. The overconfidence effect in social prediction. *Journal of Personality and Social Psychology, 4* (1990), 568–581.

11. Ebenau, R.G., and Strauss, S.H. *Software Inspection Process*. New York: McGraw-Hill, 1994.

12. Fagan, M.E. Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15*, 3 (1976), 182–211.

13. Flower, L.A., and Hayes, J.R. The dynamics of composing: making plans and juggling constraints. In L.W. Gregg and E.R. Steinberg (eds.), *Cognitive Processes in Writing*. Hillsdale, NJ: Lawrence Erlbaum, 1980, pp. 31–50.

14. Floyd, B.D.; Walls, J.; and Marr, K. Managing spreadsheet model development. *Journal of Systems Management, 46*, 1 (1995), 38–43, 68.

15. Galletta, D.F.; Abraham, D.; El Louadi, M.; Lekse, W.; Pollailis, Y.A.; and Sampler, J.L. An empirical study of spreadsheet error performance. *Journal of Accounting, Management, and Information Technology, 3*, 2 (1993), 79–95.

16. Galletta, D.F.; Hartzel, K.S.; Johnson, S.; and Joseph, J.L. An experimental study of spreadsheet presentation and error detection. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, vol. 2, January 1996, pp. 336–345.

17. Galletta, D.F.;, Hartzel, K.S.; Johnson, S.; and Joseph, J.L. Spreadsheet presentation and error detection: an experimental study. *Journal of Management Information Systems, 13*, 2 (1997), 45–63.

18. Hall, M.J.J. A risk and control oriented study of the practices of spreadsheet application developers. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, vol. 2, January 1996, pp. 364–373.

19. Hayes, J.R., and Flower, L.S. Identifying the organization of writing processes. In L. Gregg and E. Steinbert (eds.), *Cognitive Processes in Writing*. Hillsdale, NJ: Lawrence Erlbaum, 1980, pp. 3–30.

20. Healey, A.F. Proofreading errors on the word *the*: new evidence on reading units. *Journal of Experimental Psychology: Human Perception and Performance, 6*, 1 (1980), 45–57.

21. Hicks, L. NYNEX. Personal communication with the author by electronic mail, June 21, 1995.

22. Johnson, E.J. Expertise and decision under uncertainty: performance and process. In M.T.H. Chi, R. Glaser, and M.J. Farr (eds.), *The Nature of Expertise*. Hillsdale, NJ: Lawrence Erlbaum, 1988, pp. 209–228.

23. Johnson, P., and Tjahjono, D. Exploring the effectiveness of formal technical review factors with CSRS. *Proceedings of the 1997 International Conference on Software Engineering*, Boston, May 1997.

24. Lichtenstein, S., and Fischoff, B. Training for calibration. *Organizational Behavior and Human Performance, 26* (1980), 149–171.

25. Lichtenstein, S.; Fischoff, B.; and Philips, L.D. Calibration of probabilities: the state of the art to 1980. In D. Kahneman, P. Slovic, and A. Tversky (eds.), *Judgment Under Uncertainty: Heuristics and Biases*. Cambridge: Cambridge University Press, 1982, pp. 306–334.

26. MacKay, D.G. The problems of flexibility, fluency, and speed-accuracy trade-offs in skilled behavior. *Psychological Review, 89* (1982), 483–506.

27. Metcalfe, J., and Shimamura, A.P. *Metacognition: Knowing About Knowing*. Cambridge, MA: MIT Press, 1994.

28. Myers, G.J. A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM, 21*, 9 (1978), 760–768.

29. Norman, D.A. Converging industries: the multiple impacts of converging industries on politeness, the individual, society and nations. Paper presented at the Thirtieth Hawaii International Conference on System Sciences, Maui, Hawaii, January 1997.

30. Panko, R.R. A human error data website (http://panko.cba.hawaii.edu/humanerr/). Honolulu, HI: University of Hawaii, 1997.

31. Panko, R.R. Spreadsheet research (SSR) website (http://panko.cba.hawaii.edu/ssr/). Honolulu, HI: University of Hawaii, 1997.

32. Panko, R.R., and Halverson, R.P., Jr. Spreadsheets on trial: a framework for research on spreadsheet risks. *Proceedings of the Twenty-Ninth Hawaii International Conference on System Sciences*, vol. 2, January 1996, pp. 326–335.

33. Panko, R.R., and Halverson, R.P., Jr. Are two heads better than one? (at reducing errors in spreadsheet modeling?). *Office Systems Research Journal, 15*, 1 (1997), 21–32.

34. Panko, R.R., and Sprague, R.H.J. Hitting the wall: errors in developing and code-inspecting a "simple" spreadsheet model. *Decision Support Systems, 22*, 4 (1998), 337–353.

35. Plous, S. A comparison of strategies for reducing interval overconfidence in group judgments. *Journal of Applied Psychology, 80*, 4 (1995), 443–454.

36. Porter, A.; Votta, L.G., Jr.; and Basili, V.R. Comparing detection methods for software requirements inspections: a replicated experiment. *IEEE Transactions on Software Engineering, 21*, 6 (1995), 563–575.

37. Porter, A.A., and Johnson, P.M. Assessing software review meetings: results of a comparative analysis of two experimental studies. *IEEE Transactions on Software Engineering, 23*, 3 (1997), 129–145.

38. Porter, A.A.; Sly, H.P.; Toman, C.A.; and Votta, L.G. An experiment to assess the cost-benefits of code inspections in large scale software development. *IEEE Transactions on Software Engineering, 23*, 6 (1997), 329–346.

39. Porter, A.A., and Votta, L.G. An experiment to assess different defect detection methods for software requirements inspections. *Proceedings of the Sixteenth International Conference on Software Engineering*, Sorento, Italy, May 16–21, 1994, pp. 103–112.

40. Pulford, B.D., and Coleman, A.M. Overconfidence, base rates and outcome positivity/negativity of predicted events. *British Journal of Psychology, 87* (1996), 431–445.

41. Rasmussen, J. Mental procedures in real-life tasks: a case of electronics troubleshooting. *Ergonomics, 17*, 3 (1974), 293–307.

42. Rasmussen, J. The role of perception in organizing behavior. *Ergonomics, 33*, 10–11 (1990), 1185–1199.

43. Reason, J.T. *Human Error*. Cambridge: Cambridge University Press, 1990.

44. Russell, G.W. Experience with inspection in ultralarge-scale developments. *IEEE Software, 8*, 1 (1991), 25–31.

45. Shanteau, J. The psychology of experts: an alternative view. In G. Wright and F. Bolger (eds.), *Experts and Decision Support*. New York: Plenum Press, 1992, pp. 11–23.

46. Votta, L.G. Does every inspection need a meeting? *Proceedings of the ACM SIGSOFT '93 Symposium on Foundations of Software Engineering*, December 1993.

47. Wagenaar, W.A., and Keren, G.B. Does the expert know? the reliability of predictions and confidence ratings of experts. In E. Hollnagel, G. Manici, and D.D. Woods (eds.), *Intelligent Decision Support in Process Environments*. Berlin: Springer-Verlag, 1986, pp. 87–103.

48. Weller, M. Lessons from three years of inspection data. *IEEE Software, 10*, 5 (1993), 38–45.