SOCIETY OF ACTUARIES

Article from:

The Actuarial Practice Forum

February 2010

Spreadsheet Issues: Pitfalls, Best Practices, and Practical Tips

Mary Pat Campbell[1]

Contents

Published in the February 2010 issue of the *Actuarial Practice Forum*

[1] Mary Pat Campbell, FSA, MAAA, is AVP and Life Actuary at Scor in New York, N.Y. She can be reached at *marypat.campbell@gmail.com*.

Introduction

Spreadsheets are one of the largest workhorses in actuarial work and business work in general, but few of us have serious, rigorous training in their use and design. We usually pick up knowledge and skills through assignments on an as-needed basis, which can prevent a methodical approach to the technology. Alas, I have not created any such needed methodical approach, but I have found a variety of resources, general theories, and specific practices that you may wish to put into effect in your own work.

Below I assemble material I've written, originally published in *CompAct*, the newsletter for the Society of Actuaries' Technology Section. I start out with "To Err is Human; To Correct, Divine," where I highlight the general problem of spreadsheet errors and point to some useful resources. The next four sections are from a series titled "The End Users Justify the Means" ostensibly about spreadsheet design based on the end users, but the main themes are both usability as well as error-prevention. Finally, I review some presentations given at the 2009 European Spreadsheet Risks Interest Group Annual Conference, which gives one an idea the type of research and work that is being done in this realm.

I hope the ideas found herein may be useful in one's day-to-day work with spreadsheets. Most of the ideas originate with others, and what may be even more useful is the list of my original resources at the end of this paper. This paper is directed primarily toward those who create or maintain spreadsheets, but their managers can also gain valuable insights from the material presented here.

To Err is Human; To Correct, Divine

How error-riddled are your spreadsheets? How much can a simple Excel flub cost your company? When you do make a mistake, how likely are you to catch it?

Though we often work with specialized software, Excel is the central tool for most of us, easily adaptable and giving us results faster than much more complicated and targeted programs. However, the danger for material error is there, and the bad news is it's well-nigh impossible to escape error. Even worse, Excel errors can cause large financial damage: at the European Spreadsheet Risks web site, they have a page of spreadsheet error horror stories. As an example: a Canadian power company took a 24-million-dollar loss in 2003 when a cut-and-paste error led to a mispriced bid. This is not a one-off event: as of September 2007, the EUSPRIG has 89 news stories, dating back to 1995, of substantial spreadsheet errors. Minor mistakes don't show up in newspapers.

Of course, we're experts, so our spreadsheet errors are rare—right? Let's go to the research and see.

Ray Panko, a professor at the University of Hawaii and a researcher into error rates in spreadsheets, has found that in audit research of spreadsheet errors, 94 percent of spreadsheets reviewed had errors, and about 5 percent of cells in the reviewed spreadsheets contained errors. Some of these errors are immaterial, such as minor typos, but the most insidious type of errors are the ones least likely to be found: omission errors, where something is missing; and logic errors, where the model or calculation is just plain wrong. We are very unlikely to discover the errors in our own thinking (the source of logic errors), and it's very hard to see that something is not there without explicitly looking for it.

We may think, "Sure, those studies show high error rates, but that's because they're looking at the spreadsheets of a bunch of schmoes ... most likely MBA students." OK, yes, some of the research subjects were MBA students, but in controlling for expertise level, error rates were similar for novices and experts. Even when the spreadsheet task was greatly simplified, the cell error rate was 2 percent for a very artificial situation, as opposed to operational spreadsheets from real businesses. Spreadsheets used in business often were much more complicated, involving links to other files and macros doing a great deal of the calculations.

The reason for novice error rates is simple: they don't know what they're doing. But what about us experts? The problem there is we may underestimate the likelihood of error. It's hard to detect what you don't expect. If you do only the most cursory of error checks, because you are confident about what you've done, it's highly unlikely that you'll discover that you set up your model incorrectly.

Someone else may not catch the error because they don't know enough about the spreadsheet to understand when you've made a mistake. So it's a bind: other people might be more inclined to consider the possibility of an error, but they are unable to find it from ignorance; you would be able to find it, but you're overconfident about your work.

The good news is that there are ways to mitigate the errors, and to reduce your chances for error. At the end of this paper, I've provided links to resources I've found to be very helpful. The most helpful of all the articles is Philip Bewig's article "How do you know your spreadsheet is right?" If you do nothing else, check that one article out. Some of the sites and articles focus on error rates and types of error; don't discount these—if you know how material errors are most likely to occur, you're more likely to catch them or prevent them.

From reading these articles, there seem to be three main ideas that greatly reduce error rate initially and improve error detection:

• Think before you create. Plan your model structure in advance, and consider extreme values that should break your model (you can use those for testing later). Work out the logic in advance, not on the fly. This will result in better structured spreadsheets and reduce the likelihood of logic errors.
• Expect errors as you work. In lab research, when people were made aware of how common spreadsheet errors were, they were much more likely to catch their errors, especially material ones. People who expect errors examine spreadsheets more carefully, perform more stress tests, and make error-checking part of their routine. Keeping your "spreadsheet ego" in check is a must.
• Work in groups. Research has found that there is a great improvement in error detection when spreadsheets are reviewed in groups. We can be blind to our own errors but very able to see the mistakes of others. As well, different people may be apt to find different types of errors, so that in combination they improve the overall error-correction rate. Panko's research has found a statistically significant improvement in error detection when people work in groups of four (a two-thirds improvement in the error-detection rate); working in pairs did not improve detection to a significant extent.

The sources listed have even more practical and technical ideas (data validation, cell protection, named ranges, R1C1 notation, and more), but the central concept is to be mindful and to be humble.

Spreadsheets have become part of the quantitative sea we swim in—let's make sure they're our Queen Mary, and not our Titanic.

The End Users Justify the Means I: The General Public as End Users

"Make something idiot-proof, and they will build a better idiot."

I first realized I had an end user problem when I got a call from Denver, complaining that my annuity payment calculator spreadsheet wasn't working any more. As I listened, I thought: who is this person, and how did my spreadsheet end up in Denver?

Many times, spreadsheets we create for one purpose and one audience end up getting re-purposed and re-audienced. In trying to make user-friendly spreadsheets, we can end up with very unfriendly users. In a previous section of this paper, I covered the issues of preventing and correcting errors in your spreadsheets; here, I assume you have that part nailed, and are now readying to hand over your work of Excel art over to a bunch of people who will ruin it … or at the very least, will put your spreadsheets through very hard use.

My antagonism with end users is a bit over-the-top, but often spreadsheet makers and end users seem to be in a war. The makers work very hard to get the moving bits working correctly, and the end users complain about the interfaces and that it doesn't deliver what they need. While it is very important to have the technical aspects of one's spreadsheet work, it is even more important that we take user needs into consideration.

1. Of course, not all users are created equal, and some people, such as those who have to maintain the spreadsheet (which could very well be us, the spreadsheet creators), don't seem like users at all. Users will have different needs depending on what they want out of the spreadsheets, and they will have different capabilities.

This section and subsequent ones on satisfying users are not intended to be exhaustive, and are based on my own direct experience as well as research into this area. At the end of this paper are several references I've used, touching on different aspects of this subject, whether it's the visual design aspect or a systems perspective.

First up: the toughest set of end users—the General Public a.k.a. the Great Unwashed Innumerate.

Granted, actuaries are not often called upon to write spreadsheets for use by the general public, though one may come close by being asked to make spreadsheet tools for the marketing or sales departments (okay, okay, many of them are numerate…now.) This is not something to be proud about. How many times have you heard outsiders describe actuaries as subject matter experts, but horrible at non-technical communication? This is our own fault.

Actuaries are experts on many topics related to risk, and many people are ill educated in this realm—we should be out there educating people on the various risks they face and how to manage or ameliorate them. Spreadsheets are but one tool we can use in this goal of communicating with and educating the public. As an example, I created a spreadsheet (which can be found on the SOA Web site here: *http://soa.org/research/pension/research-simple-life-calculator.aspx*) to give people a better feel for longevity in retirement when most life expectancy numbers are quoted from birth (and you're only given expectancies, as opposed to percentiles as well).

A. Is a spreadsheet appropriate?
   Spreadsheets may be our general utility, but they're not always the best tool for the job. I like to think of Excel as a fancy calculator, allowing me to try different inputs to get a feel for a situation, or to let me enter information specific to my needs so as not to be distracted by irrelevant information.

   If there is no interactivity, but a simple static display of information, then there's no need

for a spreadsheet. This is especially true if you don't want anybody tampering with anything in the spreadsheet. Copy and paste into PowerPoint or Word if you just want to show a table of results, or a graph.

Tip: if you are copying a graph, make sure when you paste that you go into the *Edit > Paste Special…* menu in Word or PowerPoint, and paste the graph as a picture as opposed to an Excel object. I learned the hard way that when you blithely CTRL-C CTRL-V any graphic from Excel into Word, the entire spreadsheet is embedded within the Word document—for each graph you copy over. I crashed a few people's mailboxes by sending them my overloaded Word documents before I wised up.

If you really don't want people tampering with your information, it will be even better to get it into PDF format. Too often a graph in PowerPoint or Word can be moved to where its context is destroyed, and tables of numbers can be inadvertently—or purposefully—altered.

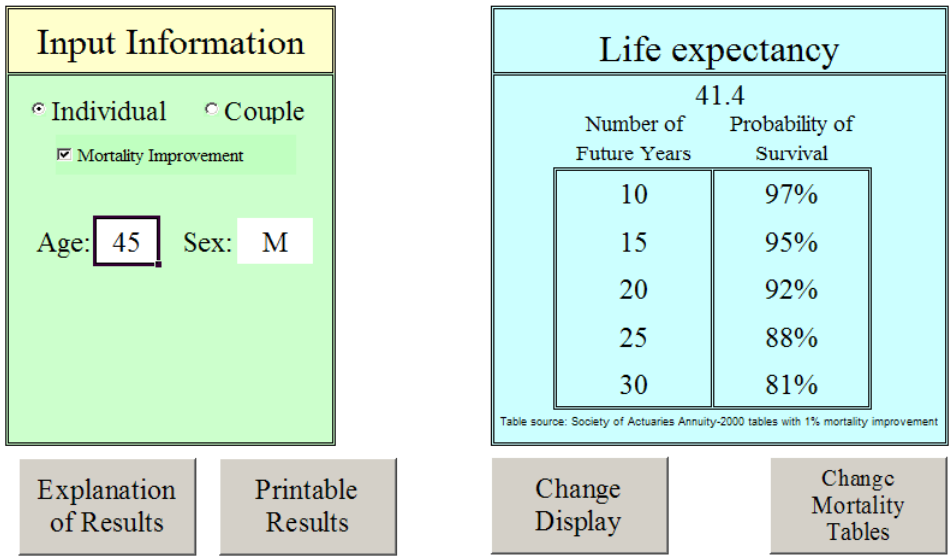
Another situation where a spreadsheet is not appropriate for a general audience is when you have sensitive or confidential information inside the spreadsheet. You may hide the sheets, lock the workbook, and password protect, but the security features of Microsoft Office are a joke. I have cracked many spreadsheet passwords with free add-ins you can find on the Internet. There is nothing secure about any Excel workbook. If you must use secret information to do your calculations, use another method, such as a Web interface to a secure server.

Tip: If you want to make a spreadsheet for general use where you strip out sensitive information from another spreadsheet, it's best to start out with an empty spreadsheet and copy over the parts you want rather than take the original spreadsheet and delete the sensitive parts. I'm probably being overly cautious here, but I can never be sure how much information Excel holds of previous versions of a spreadsheet; obviously it holds some information, or there would be no "undo" functionality. In the life expectancy calculator, the original version was for use within TIAA-CREF and had some TIAA-CREF-specific mortality assumptions. In creating a version for the SOA, I started with an empty spreadsheet and copied over only a few of the mortality tables and nothing specific to TIAA-CREF.

B. Good interface design—think iPod

Apple has been the design leader of the 21st century, in my opinion (yes, I'm a Mac person, but I promise I'm not insufferable about it. Much.) Where they have excelled is in clean, simplified interfaces, where a new user can pick up an object (such as an iPod), or open up an application (such as iPhoto), and get right to using it without having to muck about with user manuals ... and they're pretty, to boot!

When I started to design the interface for life expectancy calculator, I had Apple's design in mind. Here's a screen shot of the "front" page:



It's not the sleekest thing around, but it's very clear—large fonts for easy reading (except for the info in itty-bitty type, which is not going to be important to most general public users), clearly defined and separate input and output areas. There's no paper clip yabbering at you, asking what you'd like to do. There's a box for you to type in, buttons to push and an option to click on and off. Every button and option is clearly labeled with the least amount of words (I do think "Input Information" is clunky, though—I'd be happy to hear suggestions for my next version.)

Here are some general principles or issues I've considered in designing user interfaces for the general public:

1. Visual structure—having well-defined areas on the page for various functions, whether input, output, explanatory or other. Also, always turn off the grid and row and column labels—only designers/maintainers will want to look at that.
2. Uniformity of style—you don't want different elements clashing with each other. To go back to the Apple example, the iPod aesthetic is immediately recognizable, though it has expanded from the original model to include the Shuffle, the Nano, and the Touch—all with very different capabilities, but a commonality of design.
3. Readability—Large fonts, dark text on light backgrounds, and space around different elements help improve readability of an interface. Edit wording down as much as possible—the fewer words used, the easier it is to read and the more likely it will be read. As Joel Spolsky of *Joel on Software* proclaims: "***In fact, users don't read anything****."*—a bit of hyperbole, but a good warning to prolix designers.

4. Color—Excel has a limited default palette, which is why my spreadsheets tend to end up looking like an Easter window at Macy's. Color should be used with a specific purpose in mind. Here, I use color to set different regions apart in the spreadsheet. I used black text on a yellow background as it provides one of the most eye-catching contrasts for reading (why do you think warning signs are yellow and black?) so that it would grab visual focus when this screen is first presented.
5. Font—Some fonts are easier to read than others. They all become less readable if you mix them up or lard on too many features, such as underlining, italics, boldface, all caps, colors other than black, and so forth. Generally, I use just one font—Times New Roman, a serifed font, at a sufficiently large size. Sometimes I use the sans serif Arial (the default for Excel most of the time) if I need to write something very small; serifed fonts do not work well at small size as the text-to-space ratio approaches illegibility.

C. Hide what isn't needed

I hide a lot of things in my spreadsheet. First, I turn off sheet tabs, the grid, and row and column headers. I hide input fields when you click on particular options and hide ridiculous results. For example, I do not show probabilities for survival beyond age 95, because the mortality tables beyond 95 are unreliable. But the end user doesn't need to know that. If you click between "Individual" and "Couple," you will note that the age and sex input areas "disappear" and reappear (I do this by using VBA to set the background and font colors of the pertinent cells to the background color for the area).

If you want to see what lies behind the calculation, feel free to unlock the spreadsheet (I will talk about that below) and see what I've got floating around. Calculating life expectancy is pretty simple, so there's not much to it. You will see some features I will talk about in later articles, which should be of use to auditors, testers, and maintainers, but is of no interest to my target audience.

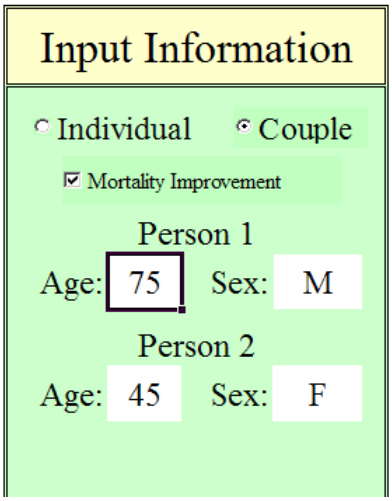
Hiding what the user doesn't need or want to see provides for a cleaner interface and reduces confusion. Also, in hiding the tabs and using buttons to force the user to navigate between two possible pages, I am controlling how the user navigates their way through the tool. I have done this many times for multi-paged spreadsheets, to make sure that all input pages are gone through before a final calculation.

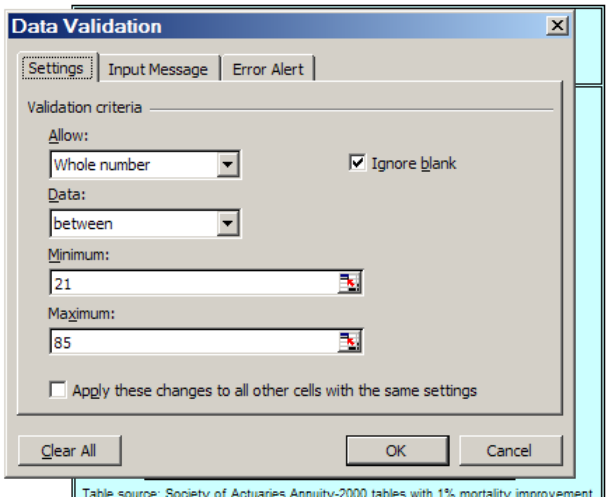
D. Constrain inputs and have sensible defaults

With the hoi polloi, one should have Data Validation on every input cell. Sure, the input prompt says "Age," but just watch—someone will type in "Forty-two" in that field instead of 42.

If you've never used Data Validation before, let me give you a quick walkthrough. I use two types of Data Validation in my spreadsheet. The first one relates to entering the age.

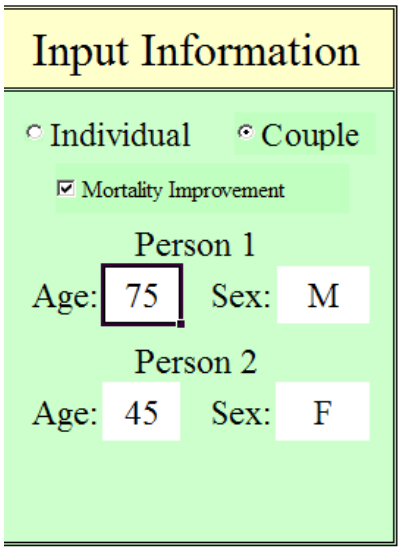
To check this out, you need to unprotect the sheet (Go to Tools > Protection > Unprotect Sheet—I don't use passwords). Highlight one of the age entry areas and pull down Data > Validation ... and you should see this:

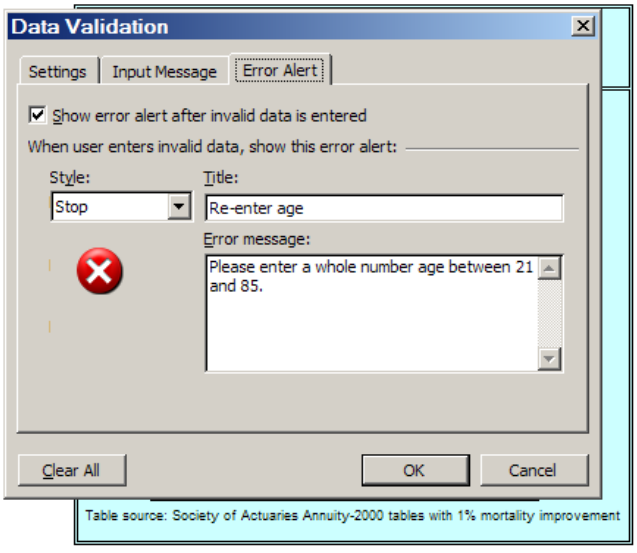




I restrict entry of age to whole numbers between 21 and 85. There are some reasons for this—first, the point is to help people gauge retirement longevity or longevity from their current age. This is not intended for children; that explains the lower bound. The upper bound is due to the scarcity of data at high ages and lack of reliability in the extreme ages. Ideally, I expect the tool to be used for people age 45–70, but I expanded it to a more reasonable range.

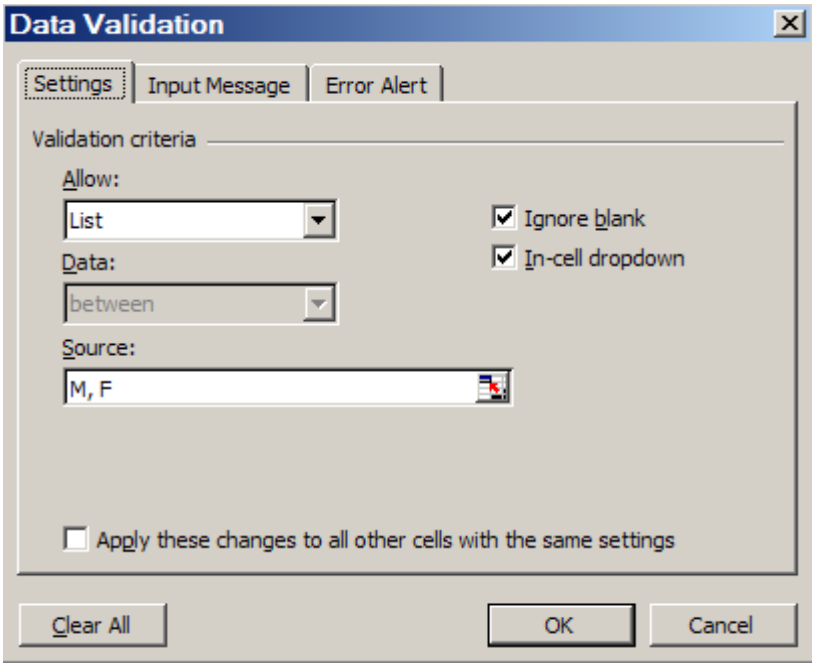
When someone tries to enter something that does not fulfill the validation criteria, they get an error message. There is a generic one, but you should never allow that. You should always customize the error message to let the user know what you are looking for in that field:





Notice there is a third tab, which calls for an Input Message. I usually keep this empty, so as not to annoy knowledgeable users. I figure I should correct the user only if they have erred. One might think an Input Message would be appropriate when the input prompt does not supply enough information as to how you want the cell filled; I say that's a failure of design and the interface should be redesigned.

The other Data Validation I use is for sex:



The "List" criterion is a very useful option in Data validation, if you have a small list of options to choose between. You can enter the list (as I did here), or refer to a range of cells (if you use cells from a different sheet, it must be a named range). The most useful feature of this option is the In-cell dropdown list, which automatically pops up when the cell is selected. The user will then automatically pick one of the items on the list. (Note that I do not take this for granted—I still specify an error message, just in case the user is unusually creative.)

While data validation is crucial to ensure that all inputs are such that calculation errors won't occur, another item to add is reasonable defaults to all input cells. I did not actually do this with my life expectancy calculator spreadsheet as the inputs are so simple. The reason to populate input cells with default values is to give the end user an idea of the range of values one is expecting to be entered. As well, many times people will not want to change defaults if they are not knowledgeable about the item entered (such as an expected interest rate, as an example).

E. Information at the point of need, only to the extent needed

Notice that I keep the explanation of what these numbers mean tucked away. I expect non-experts to need lengthy explanations and concrete examples to interpret the numerical results of my life expectancy calculator. However, I want to keep a clean interface and don't want to bother them with an explanation when they don't need it. There's a reason people got annoyed at Mr. Paperclip.

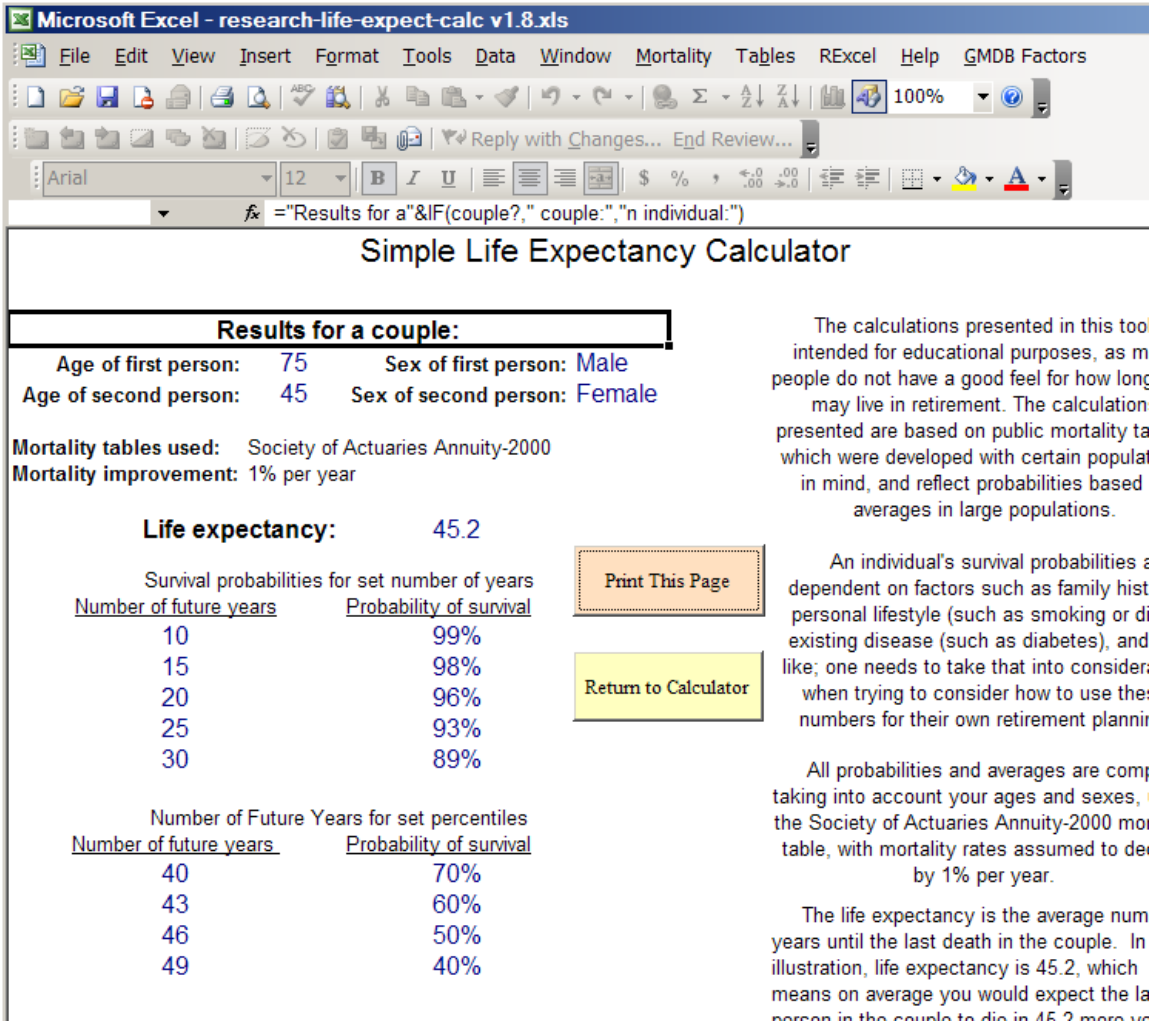
On the input page, I have a button simply labeled "Explanation of Results". If you click that, an explanation of the numbers in terms of life expectancies and survival probabilities comes up—and it's directly related to what is currently in the input fields. Once one has read the explanation once, that will probably be enough for many users, and they can go back to playing with the calculator without any distracting explanations.

Note, also, that I don't go deeply into the assumptions behind the mortality tables used (that comes later). When people are playing around with the simple interface, they just need to be able to interpret what the numbers mean. People generally don't like reading a bunch of text off a computer screen, so I try to keep the explanation as succinct as possible.

F. Printable results

The clean interface and simple explanations are all very well, but they don't print out well. A computer screen is very different from a piece of paper. The empty space and large fonts that help with interactivity become low-density information when printed out. As well, the pop-up boxes with explanations do not appear in a print out.

It is a good idea to have a printable results page that has all inputs and outputs, as well as more detailed explanations than were on the input page. Sometimes your interface design will lend itself well to both computer screen and paper, but it's unlikely. The design concerns with a printable page are somewhat different from the computer interface—you will still want to have well-defined regions, carefully pick a font, and make sure the overall design is consistent. However, you will not want to be using much color, as you can't count on your user having a color printer.



On a printable page, one can include far more results, and wordier explanations. The idea is that a person will print out the results and will be able to contemplate them at their leisure. Note still that I know the important part for most people will be the numerical results, so I used larger fonts for these items.

One should be able to immediately focus on the "meat" of the page in looking at the printed product. People (who read English, at least) focus on the top left before anything else—it's habit from decades of reading starting in that location. So that's where I put the prime information.

G. Bulletproof your code

Simply put: the user should never see an error message without instructions on how to fix the problem, and you should never allow a user to do something that breaks the spreadsheet.

This is difficult to do, as you may think you have covered every possible screw-up, only to find that the user has found a new way to screw up. Yay.

As noted above, one method of bulletproofing your spreadsheet is to add data validation to input cells. Another thing to consider (which I did not do in my own spreadsheet) is to add catch-all error handling in your VBA (get to know "On Error") and in Excel itself (get to know "ISNA" and "ISERR").

A quick tip: make sure the spreadsheet recalculates after any change in inputs. My default for my Excel is manual calculation (as I'm often dealing with large spreadsheets or volatile functions), and your user may also have such an option chosen when they open your spreadsheet. You can set the mode to automatic calculation upon the opening of the spreadsheet, but this could annoy the user (they may have good reason not to want automatic calculation), or the user may change it back to manual while the spreadsheet is open. You do not want the input and the results to be out of sync with each other

H. Own your spreadsheet and protect the original

There should be contact information in the spreadsheet itself, and clear ownership of the spreadsheet. If you did your job well in following the rules above, you shouldn't have too many people contacting you about problems. Note in my life expectancy calculator, I have my name (people might have to know to look me up in the SOA directory, but if they search for my name online, they will have an easy time finding me). In addition there's a link to the download page, which has the email of an SOA staff member, as well as a link to download the current version of the tool.

The main reason to do this is to get feedback on your spreadsheet. There may be errors or missing features desired by users that you didn't catch, and allowing for users to contact you directly (or someone else who can contact you directly) will most efficiently get this information to you.

As well, you want to protect your original spreadsheet. In the method of dissemination shown here, the original is on a web server, so the end user cannot tamper with it in any way. Sometimes, though, your spreadsheet will be sitting in a shared directory where anybody can use it. Some thoughts on handling this situation:

1. **BACKUP!** You should always have more than one copy of your spreadsheet lying around. I have my most important spreadsheets not only in a shared directory, but also my hard drive, and my personal network directory. If it's that crucial, I will even email myself a copy, to have a copy sitting on the mail server.

2. **Automatic copy:** You can write some VBA code that will check the name/location of the file to see if it's the original. If it is the original, then it automatically saves the file under a different name. Here's some sample VBA code, which would be in the code under the VBA object "ThisWorkbook":

```
Private Sub Workbook_Open()

If (ThisWorkbook.FullName = OrigName) Then
    ThisWorkbook.SaveAs Filename:=BaseName & "TEMP"
End If
End Sub
```

OrigName and BaseName are defined as constants in a module. This is just an example. I think having the spreadsheet save a new copy every time it is opened is overkill.

As I said before, these ideas are nowhere near exhaustive. These issues come up in a variety of situations, so checking out the references at the end of this paper would be a good place to start thinking about interface issues (generally they are thinking in terms of other contexts, but it still works for spreadsheets).

The End Users Justify the Means II: The Wrath of the Numerate Decision-Makers

It started out innocently enough.

In May 2008, *CFO.com* featured a Q&A session with two professors talking about spreadsheet pitfalls that their business students often fall into, with tips and warnings for spreadsheet creators.

Then the floodgates opened. The response to the article came fast and furious—though "peevish" seems to be the more appropriate description. Some choice words from respondents:

- "*Nearly every new college grad I hire has no idea how to actually create a spreadsheet that is ready to publish!"*
- "*One of the most annoying issues is the numbers not being in a common format, which makes the reading of numbers on the screen or on a printout very difficult."*
- "*How many people actually* test *their spreadsheets? With the formulas hidden, it's so easy to introduce a mistake and not even know it."*
- "*I wanted to emphasize the horror of receiving a 20-tab or greater spreadsheet that does not delineate which tabs I, as the reader, should focus on."*

So far, *CFO.com* has run two follow-up articles on their original article on spreadsheet "worst practices." (See references at the end of this paper for links to these articles.) I would recommend reading the follow-up articles, at least, as none are long, and one can get the sense of frustration from this group of spreadsheet users.

In the preceding section, I looked at designing spreadsheets for use by the general public. While certain aspects of that user group require a lot of work and thought to provide the public with a useful tool, the user group of numerate decision-makers is likely to be the most exacting audience you will ever have to deal with. From the comments seen on *CFO.com*, the expectations aren't necessarily high, but there are lots of details and features that the numerati will want.

Many actuaries are "back office" types, and their "customers" are usually the type of people who read *CFO.com* (a Web magazine geared towards financial executives). Excellence in spreadsheet design, alas, will not necessarily help in career advancement, and it is rarely taught, but adding to the frustration of the decision-makers will not have a positive impact on your position, that's for sure.

Of course, one could fill books with spreadsheet best (and worst) practices, even for just the audience I have in mind: numerate decision-makers. As with the previous section, I will look at broad principles as well as some specific recommendations. There will be some overlap, but some principles cannot be overemphasized. You can also just skip to the references—none of my ideas here are new, and you can go directly to the sources of these ideas. They will give far more details as to tricks and tips to accomplish certain actions in Excel; however, you need to think through the "why" before the "what" or "how."

A. Manage expectations, and know (and fulfill) standard expectations
   This is important in dealing with anybody in business, whether you're dealing with them through spreadsheets or not. You want to make sure it's clear to your end users as to what your spreadsheet does and, just as importantly, what it does NOT do.

   This problem is solely one of communication, and you need to put information in appropriate places so that the user will actually pay attention to the information as needed. You also want to make sure the information is where it's needed, and not tucked away in some auxiliary document that is easy to separate from the original spreadsheet.

   For example, many times we can go overboard on the list of caveats, exceptions, and all those other minutiae that detail-oriented people love so well. But people will be overwhelmed when faced with a wall of text, and may not read your huge, long list at all. Figure out what is most important, and put it permanently in the spreadsheet display

itself, or as a splash page (a window that pops up automatically when you first open the spreadsheet), or some other prominent reminder. My own preference is to pick out one bottom-line important message; e.g., "This tool is intended for exploratory purposes, so FOR THE LOVE OF ALL THINGS RIGHT AND GOOD, do not use for customer illustrations." (Excise the words in all-caps for a more professionally-worded statement.)

As for all the other piddly details, yes, put them somewhere, but hide them away. Maybe put in a button to take people to all the disclaimers, and for any kind of official printout, put them at the bottom in teensy print. If you have picked the unchanging aspects of your model appropriately, which are the sorts of things that would show up in the fine print, the user assumptions and this list should mesh. By all means, talk with the target end user before you start designing your spreadsheets, so that you know what they will expect your tool to do.

In managing expectations with regards to what the spreadsheets do, you want to make sure frustrations are low during actual operations. If you have any step in your spreadsheet operation that will appear less than instantaneous, have the spreadsheet announce it before starting the step, and if it's going to take a minute or longer, make sure you have some sort of progress bar to let people know how much has gotten done (and if you can forecast how much longer it will take, that's also useful.)

If you would like to know how to create progress indicators, I recommend any of John Walkenbach's books on VBA programming for Excel (see references).

B. Direct the flow and focus through the spreadsheets
In the responses to *CFO.com*, many people mentioned the amount of confusion when they didn't know where they were supposed to start in looking through the sheets in the workbook, couldn't distinguish results from input, and in general had a tough time knowing if they were using the spreadsheet correctly.

When designing spreadsheets for numerate decision-makers, you're more likely to have lots of input fields, multiple pages, complex output, and in general, something that one cannot reduce to an iPod-like interface, or at least not a single screen with simple inputs.

I would map out the input flow before implementing in spreadsheet format. We will see the principles I liked to use:

1. Spreadsheet always opens to introductory or first input page.
In the VBA code for the spreadsheets, you will see ThisWorkbook as one of the VBA objects, and you can put code in there (I did an example where upon opening, a new version of the workbook was saved). So you can put in code such as this:

```
Private Sub Workbook_Open()

   Sheets("Start Page").Activate
   MsgBox("This is the first input page. Contact Mary Pat" & _
    " Campbell if you have any problems.")

End Sub
```

I use the Workbook_Open() subroutine as the place to put all my workbook initializations, including setting the first page for the user to look at, any warning messages or messages as to version number or date of the last update.

2. More important assumptions are input before less important assumptions.
   This is going to work with a later principle, in that you're going to allow an out to users from any input page, if this makes sense. You have the user's attention the most at the beginning of the process, and they're more likely to flag or lose focus as they have to input more and more data.

3. At most 2 input columns per page, and at most 15 rows of input.
   The main reason is that you don't want users to miss input fields. Once you expand beyond two columns of input (and you can set up input areas pretty clearly for two columns), you are starting to get into the dangerous area of input being overlooked or being off the screen view due to the user's preferences of font size or window size. I'm assuming you're using at least four columns for two input columns, with labels for the input to the left of where they're typed in (or chosen).

   Frankly, one column is safer still, but most of the time you don't have to worry about users cutting off a second column for input. When it comes to rows, you really have to test it for the set ups most likely in use. The basic concept here is to make sure that nothing important is overlooked by the user.

4. Provide clear "Next" and "Previous" buttons, when input and results are a multi-page process.
   Usually, even with sophisticated end users, I would turn off sheet tabs and have them navigate through the input screens using "Next" and "Previous" buttons (as well as clearly identifying how many pages they had to go through before results). In addition to controlling the flow through the spreadsheets, I would also do an input check every time the "Next" button was punched, including looking for contradictions with input from previous pages.

5. Allow several exit points to results page.
   This works with prioritizing input as well, as if the input of less importance has a reasonable default set, or if it's likely the user will input that set once and for all and keep using that input set for multiple scenarios, just altering a few important items.

   As with the "Next" button I mentioned above, I would always check that all input was okay, and ready to calculate final results, when a "Go directly to Results, do not pass go, do not collect $200" button was pressed. When there was a problem, if possible, I would send them directly to the input fields that needed to be fixed or filled.

6. Results pages, within reason, should not require scrolling through screens—they should be viewable in one screen, with bottom-line results (totals, ROI, etc.) as close to the top left as possible.

   As with input page design, you want to make sure the most important information is in the part of the screen that is most likely to draw the focus of users. In the English-reading world, top left is the default start point on a page of text; obviously, in other reading conventions, such as with Japanese or Hebrew, the default focus may be different.

C. Bulletproof your models, your code, and your spreadsheets
   Just as with the general public, you want to make sure that your spreadsheets do not crash in the middle of normal operation, and in the case of abnormal operation, meaningful error messages are given and recovery of data is possible.

   Many times one cannot use traditional data validation for complex spreadsheets, but must check consistency between various input fields. As mentioned above, it's a good idea to check input reasonability throughout the input process, usually at key points, such as navigating to a new page. Give as much information as possible as to what about the input is making the model choke, so the user can fix it for themselves.

   Obviously, there will be errors the user will not be able to fix, and even though you're the one likely to fix it, it may help if you give some information as to where the error occurred (you never know, the person may have a staff member who can go into the code and fix the error—especially if you programmed the spreadsheet well). Any catchall error message should give direct contact information for the person who would fix any problems.

   As mentioned in another section, one of the most crucial steps for bulletproofing your spreadsheets is to get other people to test them. It may help if you get someone who

specializes in breaking spreadsheets, but these alpha- and beta-testers are likely to find issues that you are blind to yourself.

D. Provide flexibility, customization, and easy comparison
Numerate decision-makers like playing around with options, I've found; unlike with the general public, where I want a lot of the model nailed down, generally I've allowed every assumption to be played with when I have more sophisticated end users. Some assumptions are not easily changed, though, so sometimes I would customize the spreadsheet for use with a certain group. Where this is often true is having different input and output interfaces, where trying to satisfy all end user groups simultaneously can make for an unwieldy spreadsheet. To allow for easily-adapted spreadsheets, make sure nothing is hard-coded in terms of location—any cells being used in formulas should be referred to as named ranges.

One of the complaints amongst the *CFO.com* readers was not being able to compare two different input sets and their results, or being able to save their input without saving the entire worksheet. Allowing the saving of input *and* results is especially important for any spreadsheets that take a long time to calculate (say Monte Carlo modeling is involved).

In a recent working paper published by Harvard Business School, "How can decision making be improved?", it was noted that decisions can be improved when options can be compared head-to-head as opposed to in isolation. If your model calculates quickly, it may be a good idea to have mini-results areas on some input pages, so that one can flip through options to look at the impact of various choices. If the model is very complex, no matter how fast the spreadsheet calculates, it is a good idea to let the input be saved in its own file that can be loaded up, and especially to allow the results of two or more input sets to be compared against each other in the same display. This can be difficult to implement, but for crucial decisions, this will be well worth the effort.

E. Provide presentation-ready pages
What looks good on the screen may not look good on the printed page—or in Powerpoint or Word. I wouldn't prepare for all possible media displays (look good on a Blackberry? iPhone? Widescreen HDTV? At 4' by 8'?), but I would at the very least craft printable pages, as many people have trouble reading complex information off a computer screen (or, like me, want to make notes on the printout). It won't hurt too much to make them come back to you for special purposes, like preparing results for Powerpoint, but there's no excuse for not anticipating printing needs.

Standardize your formats. Make the styles (font, color, borders, other embellishments) reasonable, few and simple. Consider making printable pages separate from your regular

output views, and if you do so, make a prominent "Print Results" button on your results pages, so that users do not unthinkingly hit "Ctrl-P" (though you can override that, too.)

If your results pages will work well as-is for printing, make sure you set the PrintArea and default printing options for each page. If you're using sheet tabs to help identify pages, make sure that you have a header or footer identifying the page and workbook (one peeve from a *CFO.com* reader). When it comes to printed results, more information is usually better than less, as people can absorb the information on the page at their leisure, and the printed page, far separated from the original program, loses all its context without more explicit labeling.

In designing printable results pages, consider how well the pages will look in black and white as well as color. Even if you know that users will print out pages in color, they may photocopy the pages into black and white. Anything intended for printing should be distinguishable in black and white. That means, for plots, you should vary the line pattern and line weight to help distinguish when you have multiple data series in a graph. Also, do not use highlighting or light-colored fonts for emphasis, but use borders, and italicized or bold fonts, to make the text stand out.

F. Design informative graphics, and eschew plots that mislead

There have definitely been books upon books written on this topic. My first encounter with such ideas was *How to Lie with Statistics*, a decades-old book with very dated statistics but very relevant information on misleading graphs (amongst other things), especially in this era of USA Today infographics … almost all of which break the rules from this book.

Not everything requires graphics, though you may be asked to design dials, pie charts, and the like. When there are only a few numbers to compare, it is probably best to keep the numbers in table format. Pie charts are invariably misleading because they are difficult for people to really internalize and, most importantly, it's almost impossible to compare two pie charts. Several scenarios will likely be tested in using the tool. If graphs are to be compared, make sure they are on the same axes scale. Results have been misinterpreted because the visual differences were very obvious, but the plots were on entirely different scales. Some 3-D plots are also a bad idea because they are hard to interpret visually.

While graphics mislead, sometimes bare numbers can mislead more. One place where I think graphics are required is when one is showing correlations—you should have scatterplots here, and if the correlation does not satisfy the "interocular trauma test" (i.e., it hits you between the eyes), the correlation is probably bogus, just a statistical artifact.

Anyone who does heavy-duty quantitative analysis and needs guidance on how to best display the information should check out Edward Tufte's books. Tufte gives examples of best-in-show as well as horrendous graphics. *The Visual Display of Quantitative Information* is a good starting point for exploring Tufte's work.

In addition to plain graphs, one information-display technique that is in vogue amongst the numerati are dashboards. They are intended to provide easy comparisons at a glance, or keep people updated as to the state of the business. Check out the links in the references for good examples of dashboard design. When designing dashboards, you're trying to display a large amount of information in a few graphics, and trying to make it as simple as possible so that multiple lines may be compared against each other. Dashboards can be created with or without VBA, requiring connection to databases or not. As with other informational devices, you need to make sure that the information can be digested by the user, and is not too dense to be useful.

G. Own your spreadsheets, and control your versions

As with bulletproofing, this is a recommendation I'm making for spreadsheets, no matter the end user. The reason here is a little more political, though. In addition to reducing the frustration of users by letting them know who should be contacted, if you've done your job well, surely you want the decision-makers to know who you are.

As mentioned previously, excellence in spreadsheet work is rarely noted by the end user. Most people only notice technology when it doesn't do what they want it to do, and the perfect technology is that which is invisible, so if you've done your job well you probably won't hear anything from anybody. That said, spreadsheet users are voracious in their appetites, always wanting more features or new tools, and they certainly would want to contact the person who did the work the last time. You don't need to put an animated graphic of fireworks around your name; just make sure that if the user wants the information, it's easy to find, such as putting the contact information on an introductory page or having an "Email comments" button on a page.

Version control is also crucial. If your tool is going to be used by a variety of end users, you're going to have a large range of versions out there in the "user cloud." Make sure the version number is prominent in any "About" page, and as the version number is likely to mean something to you the designer, and not the user, also put a date for the version in the identifier, so it's readily apparent when someone has an out-of-date version.

I usually keep a version history in the spreadsheet itself, on a page by itself, giving the version numbers, dates, any important changes/differences between the versions, and a list of people who have worked on each version. I hide this sheet, but any user can unhide

it and look at it for themselves, so they can request a previous version if that suits their need better, and it also gives them a development history.

The End Users Justify the Means III: The Search for Problems

Ah, yes. Our friends the auditors and testers, looking for where we screwed up in our work. Our favorite end users of our spreadsheets, huh?

Fittingly, this section will be the shortest, as the goal in dealing with this particular set of end users is this:

To spend as little time as possible with them.

Time spent with an auditor is definitely not productive—as necessary as it may be—and the time spent with a tester is mainly a function of how much one has messed up one's spreadsheets. Productive time can be spent with testers, but ideally you want to hear little more than "Yeah, looks good to me."

Keep in mind that the goals of auditors and testers are different. In general, auditors are looking at static spreadsheets, making sure the inputs are appropriate, spot-checking some of the interim calculations, and making sure the outputs are reasonable and end up in their proper places either in the next spreadsheet or software package in the line, or in financial reports.

Testers, on the other hand, generally are looking at the dynamic possibilities of a spreadsheet, having several different possible input sets. In addition, some testers are not just checking that the calculations look right, but also actively try to break the spreadsheet. At least, that's how I test spreadsheets.

So let's look at some general principles in satisfying these end users, and keeping them from bugging you:

1. Clear documentation for the spreadsheet, ideally within the spreadsheet itself.
   The following points come from the article "Is This Spreadsheet a Tax Evader?" as a checklist for spreadsheet auditors in desirable documentation for a spreadsheet (or general calculation package):

   - the application's purpose, what it does and how it does it
   - any assumptions made in its design
   - what standing data constants (e.g., tax, duty, and exchange rates) are used and where they are held

- who developed it and when
- when and how it has been changed since being brought into use

The author of the article, Raymond Butler, goes on to say that "Clear instructions for use should also be present."

I have found it useful to have a single sheet within an Excel Workbook to serve as my overall documentation and version control page. There is an implementation example on page 48 of ``Spreadsheet Modelling Best Practice'' (see the Sources at the end of this paper.) I think the example given is a bit verbose and should have each version information be a single row, with the information of version number, changes made, date of change, by whom, etc. each be in separate columns. It is not as pretty as the IBM format in the reference, but it makes it easier in terms of organization, and one is unlikely to miss a field of information.

Part of the reason others and I advocate having the documentation within the spreadsheet itself, as opposed to a separate document, is that one knows the documentation is synchronized with the spreadsheet itself. It is too easy to be working on a spreadsheet and totally forget about a separate document until well after the fact, so that separate documentation is rarely up-to-date. There is little excuse when the documentation is within the spreadsheet itself.

As well, if the documentation is within the spreadsheet, you should have fewer emails from auditor or tester asking to resend applicable documentation. Remember: the goal is to spend as little time as possible on these people.

2. Inputs should be clearly defined, all constants explicitly declared, and all input visible.

For both testers and auditors, the inputs need to be even more clearly defined than with general public users or even numerate decision-makers. There may be many "inputs" those users never change, that are effectively constants. However, both auditors and testers need to know about these numbers; auditors to check accuracy or reasonability, testers perhaps to change these items to test robustness of the spreadsheet.

Sometimes one may have constants defined within VBA macros for the spreadsheet. I recommend having an initialization macro automatically writing out the constants found within the VBA code onto a specific sheet within the workbook, making the implicit explicit.

All inputs and assumptions should be visible. It is difficult to check what one can't see. Of course, any auditor or tester worth his salt should be able to unhide everything (and

yes, Excel password protection is a joke and easily crackable…. but still, it doesn't look good to be trying to hide something from a tester or auditor. If you're worried they'll mess up your work, that's what backups are for. I highly recommend using those.)

As well, it would be nice from a tester's point of view if all input fields were labeled, in terms of cell or range names. You can paste all named cells and ranges onto your documentation page. The benefit to the tester is that this helps automate the testing process; the tester can write their own macros to fill in these ranges with appropriate test sets of inputs. It can also make one's life easier if one has to maintain the spreadsheet, but more on that in the next section.

3. Outputs should be clearly defined, well-organized, and all visible…as well as reproducible.

   Not much different from the inputs section, of course. However, where one may have concentrated more on graphical outputs, and a clean interface with regards to users that would use this information to make decisions or to learn about various issues, the point here is that the actual numbers need to be visible.

   Testers and auditors both often have separate spreadsheets or software that independently calculates results for specific input sets. They are generally going to want to see the actual numbers to compare against their own calculations, as opposed to a graph. And when you are giving a wall of numbers to a person to interpret somehow, if you don't want to spend your time explaining and re-explaining which outputs are where, you should have them organized in a logical, clearly labeled way.

   Another note here: you should not have function calls that are irreproducible. You may think this an odd exhortation, but in this age of stochastic modeling (yes, some people use Excel for this purpose) one may program in a pseudorandom number generator that one does not seed. Such as Excel's RAND function. This is bad practice from many points of view, but one of the worst is that this cannot be audited. I cannot think of another type of function call that may be variable other than something related to current time. The point is that testers, auditors, and you should be able to get the exact same results from the spreadsheet using the same inputs. If this does not occur, redesign your spreadsheet.

   There's really not much more to it than that. The main point is for one's processes to be transparent and organized. The goal is not to make the inputs and outputs easily interpretable as much as easy to find, so as to make it easy to compare to what should have resulted. Nothing should be hidden, as a matter of course, though much of your work will likely go unexamined.

I did not go over how one should do a thorough audit or test of spreadsheets here. I will point to the book *Spreadsheet Check and Control*, which I reviewed in July 2008 in *CompAct*. As a spreadsheet tester, I found access to the actual VBA code, and a good programming style on the part of the creator, the most helpful in my testing endeavors—mainly because by looking at the code itself, I could see what would break it. Much more efficient than trying a whole bunch of input sets willy-nilly (though I did that for checking out calculation behavior in "normal" ranges.)

A final thought—if you know your spreadsheets are going to be audited, I recommend having a separate person, and if at all possible, two other people, to test your spreadsheets before the fateful day arrives. Because of course the best way to reduce time with auditors is to not have any mistakes in one's spreadsheets. Alas, to reduce time with the auditors, you may have to increase your time with the testers. The spreadsheet author's lot is not an easy one.

The End Users Justify the Means IV: The Journey Home

And so I come to the end of this series on spreadsheet design, concentrating on what I consider the most important set of end users: the maintainers of the spreadsheets.

I will admit, the reason I give this group primacy is from a purely selfish perspective: it's likely that the maintainers of the spreadsheets you create will be other actuaries… or yourself one year later. Nothing is more frustrating than wondering "What the heck were they thinking?" when "they" refers to one's self. Also, I would rather not be on the receiving end of Byzantine messes of Excel files should I ever have the joy of being handed your spreadsheets; perhaps I will have saved myself a lot of trouble later by writing this section.
So let's jump into it. While I generally have Excel in mind, most of the principles below belong to any spreadsheet setup, as well as computing projects in general.

1. Is a spreadsheet appropriate?

   Obviously, this question should be asked before making any spreadsheet, for any use, but we're thinking from the point of view of something that will need to be maintained—input updated, interfaces changed, techniques reprogrammed, etc. When it's a one-off use, it's not as crucial a question. So often though, those one-off, throwaway spreadsheets morph into something more permanent.

   In "Spreadsheet Modelling Best Practices," authors Nick Read and Jonathan Batson give a pro/con chart of using different software. Their article was written 10 years ago, and Excel has evolved quite a bit since then, but the question of pros and cons can be used if one considers the options. As software features change, and the resources available, in

terms of people's skills, and software and hardware already owned (or budget available for the project), it may be more helpful to make a pro/con list, considering the following dimensions:

- **How much might need to be changed in the future? And what would be changing?**
  This may be difficult to answer, as again, so many times we think we're doing a one-time task and then come to find that we have to keep doing it every month. The below dimensions may be more or less important depending on the scope of change.

  If the underlying structure won't need to change, but numbers are updated regularly, it may make sense to develop something on a less flexible platform than Excel. If the structure and tasks change a great deal, and one may need to experiment a great deal, it may not make sense to write a program in C++ from scratch.

- **Flexibility**
  The reason for Excel's popularity is that it's a general-purpose tool, with so many features, a relatively easy-to-use interface, and the ability to add on more functions as one needs. That said, this can also easily lead to a mess. It may make sense to do experiments in Excel in one's models, and then put the result that will have to be maintained in a different software form.

- **Cost**
  The cost here should be split into the development and maintenance phases. Too often people think of the development cost as the full cost, but forget the costs of updating and maintaining the project in question. It can be easier to point out the development costs (in terms of software, hardware, and personnel) than maintenance in many cases, as one might not know exactly what's involved until the dreaded maintenance comes.

  According to Barry Boehm and Victor R. Baesili, in their article "Software Defect Reduction Top 10 List," highly-dependable software tends to be more expensive in initially developing than low-dependability software—but that the operational and maintenance costs can easily swamp the "savings."

- **Development Time/Maintenance Time**
  Excel spreadsheets usually come together more quickly than other choices, unless there's a software package already set up for the type of task you're trying. The development time is a function not only of the flexibility of the software, but also the years of knowledge most actuaries have of using spreadsheets.

- **Run Time**
  There are ways to optimize Excel runs (such as turning off screen updates), but in general, if speed is what's foremost, Excel is suboptimal. I remember writing a Monte Carlo simulation in Excel about five years back. I commandeered a few workstations, set the spreadsheets to running, and then walked away. If I were doing it now, I would use R, given that I had to do a lot of experimentation. There are also software packages on the market, such as Crystal Ball, that are especially set up for this sort of thing.

- **Transparency/Complexity**
  The reason I like using Excel compared with proprietary software packages is that the stuff I want to mess with, I can. In some packages, the underlying code is too much of a black box for me. However, not everyone wants or needs this level of control. If one is using a black box-type setup for important calculations, I recommend doing stress-testing to make sure it's giving you correct (or at least reasonable) results.

- **Computational power/optimization for the particular task**
  This is a variant of some of the issues above, but the point is this: spreadsheets tend to be a general-purpose tool. There are software packages, such as R and SAS, that were developed specifically for statistical computation; database programs such as Access for slicing/categorizing large amounts of data; actuarial illustration software set up for various insurance products—while many of the same tasks can be done in Excel, they may optimally be done in other software environments.

  Of course, you may have a bunch of disparate tasks to do, such as file manipulation, data processing, heavy calculation, and visualization—then the temptation is to do it all within one software setup. I think the better path may be to modularize the task and giving each part to the most appropriate software. I have been known to program some file manipulation tasks in Perl, pull the resulting files into a C++ program to do calculations, and then take those results into Excel for graphing purposes.

  Fit the tool to the task, rather than trying to keep everything within one file. Now, one may think this is asking for more trouble, but it is easier to debug a modularized setup than a monstrosity where all parts are rammed into the same file. It also makes it easier to split a task for group programming purposes.

2. Ownership
   There needs to be a clear ownership of a spreadsheet, as well as a history of said ownership. In a corporate environment, one always needs to know who to blame (okay, not the best of motives), but more benignly, the maintainer needs to know of whom one can ask questions.

Also important, and considered in the next item, there needs to be clear ownership of the spreadsheet as one may find multiple versions of a spreadsheet flying about when there's no clear, single owner of a spreadsheet.

3. Version control
   One should have one sheet of the spreadsheet dedicated to following version control. Also, all previous versions (at least major versions) should be saved (with different filenames, indicating the versions). You never know when you have to redo a calculation, using a previous version.

   Read and Batson ("Spreadsheet Modelling Best Practices") propose the following elements of version control and documentation:

   > The documentation should include:
   >
   > - a short description of the model's purpose;
   > - who built the model;
   > - how to contact the person responsible for the model; and
   > - the model version number and when it was written.
   >
   > Depending on the model, other useful items to include on the documentation sheet are:
   >
   > - details of the data which is currently in the model;
   > - some brief instructions, describing the layout of the model or how to use it;
   > - a list of recent changes to the model; and
   > - a summary of key logical assumptions in the model.

   Now, they were writing when Excel 97 was the most recent version on the market, so the complexity of the models they had in mind may be a little lower than what people are using now. I think the general documentation (data sources, purpose of spreadsheet, etc.) should be on a separate sheet, and version control should have its own sheet.

   The details in a version control entry should be: version number, changes made from previous version, and the person or people who made the changes. Other information can be included, but those are the key items. An example is given below:

Version	Author	Date	Notes
1.0	MP Campbell	10/1/2006	Original
1.01	MP Campbell	10/15/2006	Fixed VBA code for explanations of results form
1.5	MP Campbell	1/1/2007	Added: mortality improvement, explanations of methods, new Social Security table (not that different from before), printable explanations sheet
1.6	MP Campbell	1/8/2007	Looking at projection scale G
1.7	MP Campbell	1/15/2007	Implemented projection scale G, looking at calculation comparisons
1.8	MP Campbell	1/30/2007	Removed projection scale G stuff, explanation sheet wording edited, cleaning up VBA
1.8.1	MP Campbell	2/15/2007	Fixed text areas on "Printable Explanations" sheet and ExplanationForm user form

The dates here are totally made up, but other than that, these are the kinds of notes I make on Version Control sheets. The particular version numbering is unimportant, but note that I kept "moving on" with versions, even when I did something that looked like it undid a feature I had added previously.

Sometimes I gave general notes as to what I had been doing in changing in a particular version, and sometimes I gave specific details relating to variables or named ranges. It would also be useful to put any run-time bugs discovered in such a version control sheet, which can give direction to fixes that may need to be made for future versions, and can serve as notes if one needs to revert to a previous version.

4. Documentation

   This is a more general category than version control. I have discussed documentation in the previous section: "The End Users Justify the Means III: The Search for Problems." I will expand a bit more on this topic, as maintainers will have a perspective different from testers and auditors.

   One of the key tasks of a maintainer is updating any inputs, and it may be useful to have a "Maintenance" or "Updating doc" sheet, which would indicate which cells would need to be updated within the spreadsheet. Ideally, one would auto-update any information, but the problem often is that the maintainer has no control over the location of the information needed. Once I got a call from across the country, from a user I didn't even know I had, because the spreadsheet was looking for an external file and the file system structure had changed since the last time they updated the spreadsheet.

One thing I have tried, when the updating process was fairly predictable in terms of what needed updating, I made the update sheet a matter of a checklist. Something like below:

STEP 1: Check fund parameter sets

TRUE	1.a Check fund management fees - named range "MERVector"
TRUE	1.b Check fund categories - range "FundClassVector"
TRUE	1.c Check fund margin offset - range "MarginOffsetVector"

STEP 2: Check product parameter sets

TRUE	2.a Check product GMDB design flag - range "GMDBtype"
	2.b Check partial withdrawal option flag - range "PartialWithdrawaltype"

STEP 3: Populate policy information

	3.a Clear previous policy information - macro "Policy_Info_Reset"
	3.b Paste in seriatim policy info - check with IA group
	3.c Paste in aggregate product info - check with IA group
	3.d Cross-check seriatim and agg info - ranges "GMDBcheck" and "AccValcheck"

STEP 4: Run Alternative Method

	4.a Run macro "AltMethodCalculate" - DO NOT TOUCH ANYTHING WHILE RUNNING - runtime ~1 hr
	4.b Check aggregate result - range "TotAltMeth"
	4.c Do reasonability checks - run macro "AltMethReports"

Note that I gave references to the particular named cells that needed to be checked and/or updated, as well as which macros to run.

If there are items within VBA code that would need updating, generally it's a good idea to keep all constants within a single module so they are easier to find and check.

Another thing I've found helpful is to do a guide to named ranges on a documentation page. One can paste a list of named ranges, which gives you the range names as well as the references. Even though if one names a range well, the range name is its own documentation, it's a good idea to make notes for the benefit of the maintainer so they

know what the various named ranges are used for. Providing notes on which VBA macros will refer to those named ranges is also helpful.

Other things to consider including on documentation sheets: list of macros and their use (can be done within the VBA code itself, but if those are scattered through multiple modules, it becomes unwieldy); list of sheets within the spreadsheet and uses for each sheet; key assumptions made in the models; desired features for future versions. Having these "overview" kinds of documentation helps the maintainer get the big picture of the spreadsheet, and thus their particular learning curve is greatly shortened. Given that you may be the person using this documentation, one year after you last looked at the spreadsheet, think about the kinds of information you would want to know.

Of course, in addition to the big picture, the maintainer may need the "detail" view, in that they need documentation at the level of use/computation. By this, I mean having cell comments indicating what's within a particular cell (or format conventions indicating an input cell, an intermediate calculation cell, a final result cell, etc.) and having comments within any VBA code itself.

5. Security—Do Not "Password Protect"
   The reason I put the above phrase in sarcasm quotes is that there's nothing secure about using most spreadsheets. Excel password "protection" is (relatively) easily cracked, and I've had to do it before when someone had locked a spreadsheet and subsequently left the company, or, even more annoyingly, the person forgot the password they used.

   I have nothing against "locking" spreadsheets against changes, without using a password. This will keep most people who have no business messing with locked cells and code from doing anything, and the people who know what they're doing are only momentarily annoyed.
   However, let us suppose that password protection is actually effective—this greatly complicates maintenance. Given how often people not only leave jobs, but also move around within an organization, if you have a spreadsheet that's run once a year and it's password-protected, the chances are high that the password will be forgotten. And if you have to write down the password somewhere, that's not very secure, is it?

Again, these are just some general ideas to make the task of maintaining a spreadsheet easier, and I'm sure there are many that could be added to the above list.

If you have practices that help in maintenance of spreadsheets, or other programming packages, consider sharing them with the actuarial community. Too often we are thrown into various software practices as entry-level actuarial students, and good computing practices are picked up piecemeal, if at all.

EuSpRIG Meeting Focuses on Doing It Right the First Time

This past July, EuSpRIG [European Spreadsheet Risks Interest Group] held their annual meeting in Paris, under the theme "The Role of Spreadsheets in Organizational Excellence."
From basic research in the sources of spreadsheet error to very concrete, practical tips for the daily user, to policy and auditing discussions, this year's conference adds to the already considerable resources produced by the members of EuSpRIG. Let me highlight a few of the presentations:

Keynote: Technical Standards for Modeling
This presentation was given by Deniz Sumengen, from the Board of Actuarial Standards (BAS) at the Financial Reporting Council. BAS was created after the Morris Review in the United Kingdom, and is tasked with setting up independent actuarial technical standards for that country.

In Sumengen's presentation, she highlighted an exposure draft on technical actuarial standards in modeling (TAS M), where modeling is defined very broadly, and is indeed a large part of actuarial work.
She noted there were a variety of general problems with models:

- Lack of testing
- Poor documentation
- Misunderstanding (of what the model covers, or what the results mean)
- Over-reliance on an established view
- Unrealistic assumptions

Below, I draw out some of the main points from the draft exposure of TAS M:

- Documentation shall be sufficiently detailed, include statement of purpose of the model, and be clear, unambiguous, and complete
- Models shall represent all phenomena relevant to their purpose
- Models shall be no more complex than can be justified
- Documentation shall include assumptions used in the model
- Model results shall be reproducible
- Checks will be constructed, performed, and documented to test theoretical, implementation, and end result issues
- Model limitations shall be disclosed

This is a rather robust, rigorous set of requirements. Of course, judgment on the part of the modelers plays a large role in these standards, but the principles are good for normal practice. The FRC is inviting comment, and some comment can already be seen on the current draft

report. While this would apply only to the United Kingdom, I have been told by Sumengen's colleague Louis Pryor that they invite comment from anybody.

If nothing else, look over the report (they have the comments at the beginning, and TAS M itself can be found at the end of the document), and consider incorporating these practices in your own work. I think getting into these sorts of practices will definitely help with dealing with the more complex modeling that is becoming part of the standard actuarial toolkit. See the links at the end of this paper.

Self-checks and Controls in Spreadsheets

This presentation by Patrick O'Beirne focused on very concrete practices to check one's spreadsheets. These practices are:

1. Cross foot
2. Balance
3. Proportion
4. Multiple plus ungood
5. Room for expansion
6. Other sources of information
7. Expectations
8. Top ten spreadsheet questions checklist

Let me talk about a few of these items. The first, cross foot, involves doing column sums and row sums on the same information, and making sure the overall total is equal for both. This is one of the oldest spreadsheet checks extant. He recommends having this crosscheck cell flagged with conditional formatting, so that the difference pops out to your attention if the difference is beyond a certain tolerance (the difference is unlikely to be zero, just from floating-point arithmetic issues).

The fifth item, room for expansion, relates to a common problem with formulas over ranges: what happens if you insert or delete cells in that range? Often there are issues of missed cells in sums because one has inserted new data at the beginning or end of the range (a problem, I'll note, that is caught by cross-footing). O'Beirne recommends having sums start and end with empty cells, for if you insert cells/rows/columns at the beginning or end of the ranges containing numbers, Excel will properly update.

The final item is a checklist of questions that I highly recommend. I would make an analogy to the preflight checklist pilots perform. Once you have this routine, you won't have to worry about particular issues being forgotten. Many professionals in other areas have complained about institutionalized checklists, as being demeaning of their great professionalism and intellect

(pilots originally complained, and similar systems have become part of pre-surgery in hospitals, but not without complaint), but this has been a very effective tool in reducing operational risk.

Check out the links at the end of this paper to see descriptions of the other items in O'Beirne's list. I previously reviewed O'Beirne's book *Spreadsheet Check and Control* for *CompAct*, and these items do show up there. But if you want a free, short list of tips you can apply right away—here is a link to O'Beirne's presentation: ***http://eusprig.org/self-checks,controls,pob.pdf***

An Exploratory Analysis of the Impact of Named Ranges on the Debugging Performance of Novice Users

This paper, presented by Ruth McKeever, Kevin McDaid, and Brian Bishop of the Dundalk Institute of Technology, won the Student Prize from the conference, as the judges noted it was a "well-designed and thoroughly executed piece of research."

One of the simple "good practices" in spreadsheet design has been to use named ranges as opposed to opaque references as AC4 when building formulas. That's the conventional wisdom, and the experimenters set out to investigate this, as many spreadsheet "best practices" have been developed through individual experience and common sense, but no real scientific investigation.

A small group of college students, who had been trained on spreadsheets the year previously, and who were given a little training on named ranges in Excel, were asked to debug a simple accounting spreadsheet. One group got spreadsheets using named ranges, and the other got one without.

The types of errors that had been entered ranged from non-material typos (e.g., misspelled header), rule violations (items contrary to written company policy), data entry errors (wrong numbers), and formula errors (wrong logic, wrong calculations). In their results, they found little difference between the correction rate for the first three categories, but a noticeable difference for the final category of formula errors—the most serious type of error to occur in a spreadsheet, usually, and awfully common.

Those given the spreadsheets with named ranges found *fewer* formula errors than did the control group. The researchers posited a few explanations: high cognitive load (students did not develop the spreadsheets, and would have to keep checking the names and what cells they referred to), misplaced confidence in names (would do spot check, see expected named range, and move on without seeing error), or just plain lack of understanding of the error or how to correct it. Also, some of the range names were very long, and it could have been a function of poor naming conventions.

I cannot say that I am much surprised by the results. In previous research, there has been shown different behavior of novices vs. experts when it comes to spreadsheet error and debugging. It would be interesting to see what the results were for experts, but it may require more complex spreadsheets in order to discover differences in debugging results.

There are limitations to this study, as freely noted by the researchers themselves, but it points out the important lesson that we should put our assumptions of risk management techniques (here, reducing spreadsheet error, as an operational risk) to the test. See the link at the end of this paper.

For more papers and presentations from the EuSpRIG conference, check out the group's Web site *http://eusprig.org/*. You can find capsule reviews of the presentations at Patrick O'Beirne's site: *http://www.sysmod.com/praxis/prax0907.htm*.

You can find the research papers at the archive site *http://arXiv.org* using the search term "eusprig," which will bring up this year's papers as well as papers from previous conferences.

Closing Thoughts

I hope you can use the above items as a jumping-off point for your own work with spreadsheets, as well as provide incentive for others to write about the pitfalls, best practices, and tips they've developed over the years. Too often, we back office types toil in anonymity, and information-sharing can really improve our "game."

When I entered the corporate world, I had never even used Excel before, and like many others, I taught myself from reading books (I highly recommend those from Walkenbach) and from dealing with other people's spreadsheets. Most of us never get any formal training in how to use one of the central tools of our modeling analysis; this is not unique to Excel and spreadsheets, but common amongst most of the numerical computing tools we use.

While this paper is neither exhaustive or definitive, I see it as a part of raising awareness for these issues, and I hope to see more materials in the future in providing better education for actuaries with regards to numerical computing.

References by Section

To Err is Human; To Correct, Divine

Philip L Bewig, "How do you know your spreadsheet is right? Principles, Techniques and Practice of Spreadsheet Style," 2005. http://www.eusprig.org/hdykysir.pdf.

Theo Callahan, "Block That Spreadsheet Error," *Journal of Accountancy* (August 2002). http://www.journalofaccountancy.com/Issues/2002/Aug/BlockThatSpreadsheetError.htm.

European Spreadsheet Risks Interest Group, http://www.eusprig.org/.

Ray Panko's Spreadsheet Research, http://panko.shidler.hawaii.edu/SSR/index.htm.

Spreadsheet error news stories from EuSpRIG, http://www.eusprig.org/stories.htm.

Ray Panko, "What We Know About Spreadsheet Errors," January 2005. http://panko.shidler.hawaii.edu/SSR/Mypapers/whatknow.htm.

Tuck School at Dartmouth: Spreadsheet Engineering Research Project, http://mba.tuck.dartmouth.edu/spreadsheet/product_pubs.html.

Systems Modelling Ltd., Spreadsheet Research Resources http://www.sysmod.com/sslinks.htm#Research.

The End Users Justify the Means I: The General Public as End User

Joel on Software, online series on user interface design for programmers. Starts with this post: http://www.joelonsoftware.com/uibook/chapters/fog0000000057.html.

Wikipedia, "Typography," subsection on Readability and Legibility, http://en.wikipedia.org/wiki/Typography#Readability_and_legibility.

Color rules of thumb, http://www.writedesignonline.com/resources/design/rules/color.html.

Jakob Nielsen, "Ten Usability Heuristics," http://www.useit.com/papers/heuristic/heuristic_list.html. (Jakob Nielsen is a usability guru—I highly recommend his publications, http://www.useit.com/jakob/)

Bruce Tognazzini, "First Principles of Interaction Design," 2003. http://www.asktog.com/basics/firstPrinciples.html

The End Users Justify the Means II: The Wrath of Numerate Decision-Makers

Ansari, Shahid and Block, Richard. 2008. Spreadsheet Worst Practices. *CFO,* May 14. http://www.cfo.com/article.cfm/11288290.

CFO.com Staff. 2008. Sloppy Spreadsheets: Readers Speak Out. *CFO,* June 18. http://www.cfo.com/article.cfm/11525407.

Harris, Roy. 2008. Sloppier Spreadsheets: How Bad Can They Get? *CFO,* August 20. http://www.cfo.com/article.cfm/11950766.

Edward Tuft's home page: http://www.edwardtufte.com/tufte/.

Edward Tufte Writings: http://www.edwardtufte.com/tufte/newet.

"Ask ET" (Q&A with Edward Tufte), http://www.edwardtufte.com/bboard/q-and-a?topic_id=1.

Excel User dashboard info http://www.exceluser.com/dash/index.htm.

Jorge Camoes, "How to create a dashboard in Excel," http://charts.jorgecamoes.com/how-to-create-an-excel-dashboard/.

Milkman, Katherine L., Chugh, Dolly and Bazerman, Max H. "How Can Decision Making Be Improved?" 2008. http://www.hbs.edu/research/pdf/08-102.pdf.

John Walkenbach's Web site http://spreadsheetpage.com/.

David Hawley and Raina Hawley. 2004. *Excel Hacks.* California: O'Reilly Media, Inc. Also online at http://oreilly.com/catalog/9780596006259/.

The End Users Justify the Means III: The Search for Problems

Raymond J Butler, "Is This Spreadsheet a Tax Evader ?" *Proceedings of the 33rd Hawaii International Conference on System Sciences* (2000). http://www.eusprig.org/hicss33-butler-evader.pdf.

Campbell, Mary Pat. "Spreadsheet Check and Control: A Review." *CompAct* (July 2008): 13-14. http://soa.org/library/newsletters/compact/2008/july/com-2008-iss28.pdf.

Read, Nick and Batson, Jonathan. "Spreadsheet Modelling Best Practice." April 1999.

Institute of Chartered Accountants for England and Wales. http://eusprig.org/smbp.pdf.

European Spreadsheet Risks Interest Group, "EuSpRIG Original Horror Stories" (page of news stories of spreadsheets with costly mistakes). http://eusprig.org/stories.htm.

The End Users Justify the Means IV: The Journey Home

Banham, Russ. 2008. "Up and Away." *CFO*, December 1.
http://www.cfo.com/article.cfm/12665848

Boehm, Barry and Basili, Victor R. "Software Defect Reduction Top 10 List." *Software Management* (January 2001): 135-137.
http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf

Califf, Howard. "Spreadsheets and Specifications." *CompAct* (October 2007): 4-6.
http://soa.org/library/newsletters/compact/2007/october/csn-0710.pdf

European Spreadsheet Risks Interest Group. http://eusprig.org/

O'Beirne, Patrick. 2005. *Spreadsheet Check and Control*. Ireland: Systems Publishing.

Read, Nick and Batson, Jonathan. 1999. "Spreadsheet Modelling Best Practices" Institute of Chartered Accountants for England and Wales, http://www.eusprig.org/smbp.pdf.

EuSpRIG Meeting Focuses on Doing It Right the First Time

Sumengen, Deniz. 2009, "Technical Standards for Modelling," Presentation to EuSpRIG 2009 Conference, http://eusprig.org/Sumengen,FRC,Modelling.pdf.

Board for Actuarial Standards. May 2009. "Exposure Draft: Modelling,"
http://www.frc.org.uk/images/uploaded/documents/Modelling%20ED%20Final.pdf.

O'Beirne Patrick. 2009. "Presentation to Tenth Annual EuSpRIG Conference,"
http://eusprig.org/self-checks,controls,pob.pdf.

O'Beirne, Patrick. 2009. "Checks and Controls in Spreadsheets." EuSpRIG 2009 Conference,
http://arxiv.org/ftp/arxiv/papers/0908/0908.1186.pdf.

Ruth McKeever, Kevin McDaid and Brian Bishop. 2009. "An Exploratory Analysis of the Impact of Named Ranges on the Debugging Performance of Novice Users." EuSpRIG 2009 Conference, http://arxiv.org/ftp/arxiv/papers/0908/0908.0935.pdf.

Patrick O'Beirne. *PraxIS* newsletter, (July 2009) http://www.sysmod.com/praxis/prax0907.htm.