The End Users Justify the Means: IV The Journey Home

By Mary Pat Campbell

Mary Pat Campbell, FSA, MAAA, is a vice president at The Infinite Actuary. She can be contacted at marypat.campbell@gmail.com.

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: the likely 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 article. (I highly recommend other people with similar motives also write for CompAct. Nothing motivates like self-interest.)

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 … but so often 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 paper 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 along with 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 soft-

ware 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 with 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 … pretty much, 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 give 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 (OK, 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'll 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;

CONTINUED ON **PAGE 22**

- 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 are 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.) as well as version control be on a separate 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 in the chart below (left).

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 in the chart 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 forward with versions, even when I did something that looked like it undid a feature I had added previously.

Note that sometimes I gave general notes as to what I had been doing in changing 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 article in this series, "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 checklist. Something like the chart on page 23.

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 check and find.

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

CONTINUED ON **PAGE 24**

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.

I can be contacted at *marypat.campbell@gmail.com.*

REFERENCES:

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

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

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

Campbell, Mary Pat. "The End Users Justify the Means III: The Search for Problems," CompAct, April 2009, *http://soa.org/library/newsletters/compact/2009/april/com-2009-iss31.pdf*

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

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

Read, Nick and Batson, Jonathan. "Spreadsheet Modelling Best Practices," April 1999 *http://www.eusprig.org/smbp.pdf* ■