EuSpRIG
European Spreadsheet Risks Interest Group

VisiCalc: Design Tradeoffs in the First Electronic Spreadsheet

After Dinner Speech, Magdalensberg, Austria, Thursday 15th July 2004

Daniel H. Fylstra
Founder, President and CEO, Frontline Systems Inc, Incline Village, Nevada, USA
daniel@solver.com

Dan Fylstra was the founder of Personal Software in 1978 (later renamed VisiCorp) – the firm that published VisiCalc, the original electronic spreadsheet program. Dan was deeply involved in the development and marketing of VisiCalc, from the earliest prototype created by Dan Bricklin on an Apple II he borrowed from Dan Fylstra, to the design sessions in Bob Frankston's attic, through its heyday on various early personal computers including the IBM PC. Today, Dan is the founder and president of Frontline Systems, developers of the Solver in Microsoft Excel, Lotus 1-2-3 and Quattro Pro, and sponsors of this EuSpRIG conference. He has an abiding interest in what is right and wrong with spreadsheets.

Thank you – I hope everyone is enjoying and learning from this EuSpRIG conference as much as I am. Tonight I'd like to reflect on the design of the first electronic spreadsheet program, VisiCalc, in light of what we know today about errors in users' spreadsheet models and other human cognitive activities, and the attendant risks of spreadsheet use. Many people have commented that the very features of the spreadsheet that make it easy to use also make it prone to error, that spreadsheeters are programming without realizing it, and that techniques from professional software development should be applied to spreadsheet design. I'd like to recount some of the design tradeoffs that made this so, comment on why the spreadsheet paradigm invented with VisiCalc has endured for so long, despite its known deficiencies and the risks of spreadsheet errors, and draw some conclusions that may challenge you, or at least provoke some thinking and further discussion.

To do this, I have to take you back in time – a long time ago in a galaxy far, far away … This was in fact our world, the world of computing in 1979, but it was so different from today's world that it seems like another planet altogether. The most-used computers were IBM 370 mainframes and DEC minicomputers, used mostly by large corporations and well-endowed universities and research laboratories. Computer programs were written by professional programmers, writing in COBOL, FORTRAN and PL/I. C and Pascal had just been invented; C++ and Modula lay well in the future; object oriented programming ideas were emerging in the lab, but were very far from the mainstream.

Most important, amateur or casual programmers hardly existed; applications were limited by the cost of both hardware and software. 'Personal computing' was a vision, not a reality – the first PCs such as the MITS Altair and the KIM-1 were programmed by 'hobbyists' who were, for the most part, professional programmers and engineers in their day jobs. I still vividly remember meeting my future wife, Hilary Huttner Fylstra, on a moonlit night in May 1977, and showing her my KIM-1 single-board microcomputer in my Harvard Business School dorm room. In my second year at HBS in 1978, my classmates gave me what they called the "Dan Fylstra Computerized Universe" award, as a joke – they were amused by my frequent assertions that, someday, they would all have personal computers on their desks at work, and perhaps even at home. If you asked anyone whether using computer, let alone programming, was an important skill for a future business manager, the prevailing answer would certainly have been 'no.'

In 1979, most planning and budgeting was done by hand, using a calculator and large accounting ledger sheets – the original spreadsheets. Now, the idea of using a computer for financial modeling, budgeting, planning and forecasting was not new. But it was largely the province of leading Fortune 1000 companies, who could afford professional analysts who spent their time building planning models. The leading software systems – which I viewed as the major competition for VisiCalc – were tools offered on timesharing systems, such as FINPLAN, SIMPLAN and PLAN. I'm curious – is there anyone here who worked with these systems? It's interesting to me that these systems shared some of the design features that have recently been proposed by many people as a 'cure' for the ills of electronic spreadsheets. They were declarative languages, with user-defined variable names, assignment statements, language constructs to spread calculations across time, and strong abilities to create and print or display ledger-style reports.

Into this world came VisiCalc, starting with a single Apple II computer, that I purchased directly from Steve Jobs at a negotiated discount, and a gleam in Dan Bricklin's eye. Let me recall some details of the Apple II, because it will help us understand the design tradeoffs that went into VisiCalc. It was a breakthrough in terms of computing power for the money in its day, but it was very primitive by today's standards. It used a 6502 microprocessor – based on some research, I estimate that it had about one ten-thousandth of the CPU speed of a modern 3 GHz Pentium 4. So for example, if you have a calculation that executes in 1 second on today's PCs, that same calculation would take about two and a half hours on an Apple II. A typical Apple II had 16K bytes of RAM when we started developing VisiCalc – that was our target for the program, data and operating system memory – but in the next year that increased to 32K bytes for most new Apple IIs, which helped us, because the first commercial VisiCalc program took about 20K. That's about one twenty-thousandth of the 512MB RAM on most PCs today. The screen had 25 lines of 40 characters each; the keyboard had upper case only, just two arrow keys, and no interrupts – so it was easy to lose user keystrokes. Memory was so tight that the 2000 bytes we thought we'd need for fully descriptive menu names and basic Help was an unaffordable luxury – the bytes had to go for low-level disk I/O and keyboard polling routines. It was amazing really, and a tribute to both Dan Bricklin's design and Bob Frankston's programming, that VisiCalc worked as smoothly as it did on the Apple II.

But I think it's fair to say that VisiCalc was consciously designed with an explicit awareness that we were creating an end-user programming tool, and that several of the key features "for better or worse" of the spreadsheet metaphor were deliberate design tradeoffs, made with an understanding, for that day, of the implications of the tradeoffs. All three of Bricklin, Frankston and I had BS degrees from MIT in Course VI-3, the Institute's computer science major, and two of us had MBAs from Harvard which gave us some relevant application domain knowledge.

The most important factor that's hard to appreciate from our vantage point today was that, in 1979, 'casual programming' was almost an oxymoron – only a handful of people could be said to have this kind of application development skill. Today, there are at least 100 million copies of Microsoft Excel in users' hands, and a significant fraction of those users are doing 'casual programming.' My point is that many of the things we worry about people doing with spreadsheets today – making errors, being overconfident, and so on – didn't arise on computers because there were no 'casual programmers' – but they likely had counterparts in errors made by hand, with a calculator, pencil and eraser, or by making decisions based literally on back-of-the-envelope calculations.

So the audience of potential users we were trying to address in 1979 had experience with a TI or HP calculator, pencil and eraser, and large accounting ledger sheets with columns and rows. VisiCalc created a visual metaphor for these things on the computer screen. Indeed, VisiCalc – the name that I coined – was a contraction of 'Visible Calculator,' and Dan Bricklin's earlier working name for the program was CalcuLedger. A key difference between VisiCalc and systems like FINPLAN, SIMPLAN and PLAN was that ideas or features that were separate and abstract in the other systems were unified and made concrete in VisiCalc.

Let me talk about three design features in VisiCalc, all of which had tradeoffs: formulas behind cells, A1 notation, and the Copy or Replicate facility.

The first and, I think, most fundamental idea in VisiCalc was to merge the report or display of numbers with the formulas that calculated the numbers. Systems like FINPLAN, SIMPLAN and PLAN had variables with user-defined names, arrays and subscripts to handle multiple time periods and the like, high-level assignment statements to calculate results, and commands to easily create reports for printing or display on the screen. All these concepts – variables, arrays be filled with calculations, then used in reports – were abstract. And people who were good at abstract thinking could picture these things in their minds' eyes and work with these tools. But by some estimates, 90% of human users are *not* good at abstract thinking.

VisiCalc took these concepts or features, unified them, and made them concrete, with cells on the screen that both contained formulas and displayed formatted numeric results. Bob Frankston has written about how VisiCalc created the illusion that the cells in the report and the variables being calculated were the same thing, and were completely under the user's control. Users who were familiar with a calculator, pencil and ledger sheet saw VisiCalc as a spreadsheet that behaved like a grid of TI calculator LCD displays. Now, this whole approach violated the idea of separating the model from the data. But for most users in this era, the abstraction of a model separate from the data was exactly what they *didn't* need. They needed something concrete. And that is still true for the majority of 'casual programmers' today – one reason why the spreadsheet metaphor has endured.

A second decision, and design tradeoff, in VisiCalc was the use of A1 notation for cell names or addresses. This created a default naming scheme for all of the variables that the user could create and reference in formulas. We did experiment with the alternative of numbers for both rows and columns, but concluded that A1 notation was better for most users. Years later, Microsoft would try to 'improve' on VisiCalc's A1 notation with R1C1 notation in both MultiPlan and Excel; this coordinate system went from being the only choice, to the default choice with A1 notation as a backup, to A1 notation becoming the standard in successive releases of these spreadsheets.

You might be interested to know that the original pre-release VisiCalc in 1979 allowed the user to edit the row and column labels, substituting user-defined names for A, B, C and 1, 2, 3. But this facility was turned off before the first commercial version was delivered; Bricklin and Frankston made the judgment call that "stable naming," where everyone looking at spreadsheet formulas would understand the cell references, was more important than the documentation value of user-defined names. Years later, Lotus and Microsoft introduced user-defined names as a separate facility; after decades of spreadsheeting, these 'more readable' user-defined names still are not widely used.

A third key design innovation and tradeoff in VisiCalc was the Replicate command, used to copy formulas and adjust their cell references. Lotus and Microsoft later turned this into the 'Copy' command, for better or worse. But the key idea behind the Replicate command was that it made looping, or iteration, concrete. Where a programmer or SIMPLAN user had to be able to imagine, abstractly, how a loop or multi-element assignment statement would affect the values of many cells each time it was performed, VisiCalc users could see it happen on the screen – the loop was unrolled, and the iteration was performed before their eyes. The result of the Replicate command was intentionally made identical to the alternative of typing all of the replicated formulas by hand. And the tradeoff of this choice was that the iteration and the array structure implied by the Replicate command was not 'remembered' in the spreadsheet, other than in the pattern of cells left behind. But the silver lining behind this side of the tradeoff was that it created opportunities for a doctoral thesis by Markus Clermont, and efforts by Operis and others, to find clever ways to infer the structure based only on the trace of the cells left behind.

My message is that the VisiCalc designers recognized the tradeoffs in each of these three areas, and made these design decisions deliberately, for better or worse – though I think mostly for the better. Because in 1979, many experts would have said that the term 'end user computing' was an oxymoron – end users just didn't do computing, and they probably shouldn't. Businesses were very skeptical about personal computers; those who knew about them at all often regarded them as toys. Many felt that computers should be programmed by professionals, while end users should mind their manners and use what they were given. VisiCalc and the Apple II had to overcome this major barrier, and they did. In the 1960s and 70s when I grew up, lots of students demonstrated – they took over the MIT administrative offices in my freshmen year – and they chanted 'power to the people.' But it was the engineers and geeks like Dan, Bob and I who went out and actually gave power to the people, at least those with analytical needs in business.

Finally, I would like to comment briefly on the fact that the basic spreadsheet metaphor introduced by VisiCalc has endured for more than 25 years – up to the present time. The ideas we've just discussed, and many others, from inserting and deleting rows and columns to splitting the screen, freezing header rows and columns, and synchronized scrolling, were all present in the original VisiCalc. And very few if any new ideas for the basic visual metaphor have emerged and persisted to the present time. This was not for lack of trying, at least from the early 1980s to the early 1990s: A variety of new spreadsheet designs appeared in products like Context MBA, Javelin and Lotus Improv, several with significant marketing support, thanks to venture capital or corporate sponsorship – but as businesses moved to adopt electronic spreadsheets en masse during the 80s and 90s, buyers and users consistently chose the 'traditional spreadsheet' over these alternatives, several of which were arguably better, especially from the standpoint of more structured design, documentation, and ease of modification of large models. I believe that the reasons for this have much to do with the market dynamics of technology adoption and economic switching costs, more than the design merits of the products – but that's another discussion that would take more time than we have tonight.

To conclude, there were reasons – many of them sound, I think – for the design choices and tradeoffs that led to the first electronic spreadsheet and to its success in the marketplace. It's certainly true that the balance underlying the tradeoffs between the ease of getting started and the concreteness of small spreadsheets, versus the ease of maintenance and understandability of very large models, has changed significantly in 25 years. The enormous advances in CPUs and memory have made spreadsheet models larger than anything we could have imagined in 1979 not only possible, but common. But I think we always have to bear in mind that most users compare spreadsheet models, not to the alternative of well-structured models developed by professionals skilled in working with abstractions, but to the alternative of manual methods or 'doing without' – because the ability to work with abstractions and create structured designs are still in short supply, compared to the demand for modeling applications. I suppose I've simply underlined what you already know – that it's a real challenge to persuade people to change their ways, or to design to a better metaphor than the electronic spreadsheet that large numbers of users will accept. I want to wish all of you – all of us – success in the pursuit of better productivity and reduced risks in spreadsheet applications.