

Smart Sheets

A new breed of spreadsheet computing

Lakebolt Research

A Brief History of Spreadsheets

- Pre-cambrian
 - LANPAR
 - Autoplan
- Cambrian explosion
 - Visicalc
 - SuperCalc
 - Quattro Pro
 - Lotus 1-2-3
 - MS-Excel
- The great extinction
- Cloud era
 - Microsoft Excel 360
 - Google Spreadsheets
 - ZOHO

ITEM	NO.	UNIT	COST
WHEEL RAKE	43	12.95	555.35
BUZZ CUT	15	6.75	101.25
TOP TONER	250	49.95	12487.50
EYE SHUFF	2	4.95	9.90
SUBTOTAL			13155.50
9.75% TAX			1282.50
TOTAL			14438.15

Spreadsheets

- Spreadsheets are a very popular productivity tool
- As stated by Simon Peyton Jones, the language of MS Excel formulas is the most popular programming language in the world.
- Indeed, spreadsheets support a functional (albeit limited) programming paradigm, wrapped in an intuitive visual computing environment.
- Why are they so successful?
 - Low “barrier to entry”
 - Visual and intuitive
 - Functional
 - Reactive

Current State of the Art

- Spreadsheets are credited as the killer app that helped spur the PC era
- Decades later, they are still prevalent, on the desktop as well as the cloud
 - MS-Excel
 - Google Spreadsheets
 - Zoho
- MS-Excel is the de-facto “reference implementation”
- The cloud revolution could have been an opportunity to disrupt spreadsheets
- There is a widespread acceptance of basic design assumptions
- Everyone tries to be “just like MS-Excel”
- But, is the underlying design a good one?
 - WARNING: repetitive (boring) slides ahead!
 - Quick survey of rants about spreadsheets from the Web

Issues:

from the Web

- Source:  ANKHOR
find your data flow
- **Problem 1: Difficult inspection / lack of traceability:** In complex spreadsheets it is very hard to keep the overview concerning the formulas. Auditing the correctness of a spreadsheet can become a cumbersome task, especially in documents with several sheets making cross-references all over the place.
- **Problem 2: Inflexibility:** It is hard to rearrange the solution when it turns out that the trial-and-error development process went into the wrong direction and needs to be corrected.
- **Problem 3: Limited reuse:** Applying a solution to slightly different source data sets is difficult. A spreadsheet created for one purpose can in most cases not be reused for even a slightly different problem. It is not possible to create libraries with subsets of a solution and rearrange or recombine those easily for application to a new problem.
- **Problem 4: Limited scalability:** Processing very large amounts of data in a spreadsheet causes problems as the complete source data has to be present at once – there is no simple way for “streaming” large amounts of data through the spreadsheet solution.
- **Problem 5: Deployment:** It is not easily possible to turn a spreadsheet into a standalone solution that can be deployed as a distributable product or be published as a web site.

Issues:

from the Web

- Source: 
- Susceptibility to trivial manual errors
 - Due to the fundamental structure of spreadsheets, a slight change in the formula or value in any of their inhabited cells may already affect their overall output: e.g. accidental copy-paste, erroneous range selection, incorrect data input or unintentional deletion of a character, cell, range, column, or row
- Possibility of the user working on the wrong version
 - Since the most common reports are usually generated on a monthly basis, users tend to store them using file/directory names variations. A user can accidentally work on the wrong version.
- Prone to inconsistent company-wide reporting
- Often defenseless against unauthorized access
- Highly vulnerable to fraud
- Spreadsheet risk mitigation solutions may not suffice
 - No clear ownership of risk management responsibilities
- **To get rid of spreadsheet risks, you'll have to get rid of spreadsheets altogether**

Issues:

from the Web

- Source: 
- 1. Vulnerable to fraud
- 2. Susceptible to trivial human errors
- 3. Difficult to troubleshoot or test
- 4. Obstructive to regulatory compliance
- 5. Unfit for agile business practices
- 6. Not designed for collaborative work
- 7. Hard to consolidate
- 8. Incapable of supporting quick decision making
- 9. Unsuitable for business continuity
- 10. Scales poorly

Issues:

from the Web

- Source: 
- 1) **Spreadsheet data can be erroneous when entered or changed.** Now, you can argue that this is true of all applications that involve data entry, but spreadsheets are particularly prone to this problem because the inevitable errors are hard to identify and trace. Controlling who and when changes can be made is very problematic as well.
- 2) **Spreadsheet consolidations and aggregations are not for the weak of heart.** Let's consider a marketing budget developed with a spreadsheet where marketing budget and spend by marketing communications, lead generation, event marketing and web marketing are broken out into tabs and allocated to five lines of business or geographies. That's 20 individual spreadsheet matrices that must be aggregated every time a change is made or a current report requested. One change to any of the cells requires re-aggregation of all the matrices. Yes, you can program, emphasis on the word *program*, macros to automate the process, but honestly, it hurts just thinking about it.
- 3) **Spreadsheets are not scalable.** I have seen marketing budgets that encompass hundreds of marketing programs, multiple lines of business or geographies across multiple product lines. If you are tracking and rolling up all marketing spend transactions, you can easily get into tens of thousands of lines of spreadsheet rows making reporting and consolidations slow and difficult.
- 4) **Spreadsheets are in reality not that flexible.** Using the same scenario cited in the consolidations and scalability examples described above, imagine the impact of a decision to add a column, track a new element, add a new category of spending or a new line of business. The impact ripples throughout your spreadsheet based marketing budget. Also, watch out for data entry errors when these changes happen too!!!
- 5) **Spreadsheets are problematic to inspect and audit.** We've all seen it. Your spreadsheet doesn't cross foot or total up to what's in the submitted budget or spending report. Consider a budget that was fine on Monday but is now not footing on Tuesday. Who changed it and where was the change made. What's your audit trail?

Issues:

from the Web

The Three Reasons Investors Want You Off Spreadsheets

Presented by:

Larry Tabb, Founder & CEO, TABB Group

Dr. Lance Smith, CEO, Imagine Software

Gregory Cozza, Business Development, Imagine Software

[Home](#) » [Infrared & Filtered Software Blog](#) » [Workflow Services](#) » [E-forms – how to get away from Excel](#)

E-forms – how to get away from Excel

January 22nd, 2009 by robertr
[Go to comments](#) | [Leave a comment](#)

Look beyond your spreadsheets

See the big picture instead with analytics
by Ritu Jain, Global Marketing Manager, SAS

Let's take different perspectives

- **Programming language design perspective:**
- **Data Representation**
 - Which objects can be created?
 - Which ones can be named?
 - What kind of type system is implied?
 - How can users combine existing constructs?
- **Functional abstraction and composition**
 - How can users define and combine new functions?
- **UI software design perspective:**
 - How can users present their model?
 - How much control they have over the layout?
 - How easy is to adapt to model growth and changes?

Conceptual Simplicity
Expressive Power
Orthogonality, Composability, ...

Orthogonality, Composability, ...

PI: overly simplistic state modeling

- Current Paradigm
 - State modeled as a set of named worksheets
 - Each worksheet is a 2D grid of cells
 - Cells contain formulas or values (a largely artificial distinction)
 - Cells and ranges are designated by coordinates
 - Unitype type system: every cell is a variant
- Naming of cells (and ranges) is a difficult matter
- Resulting nightmares
 - AI-soup ... programming by coordinates
 - Hard to read, understand, and validate
 - Easy to break

P2:

lack of functional abstraction

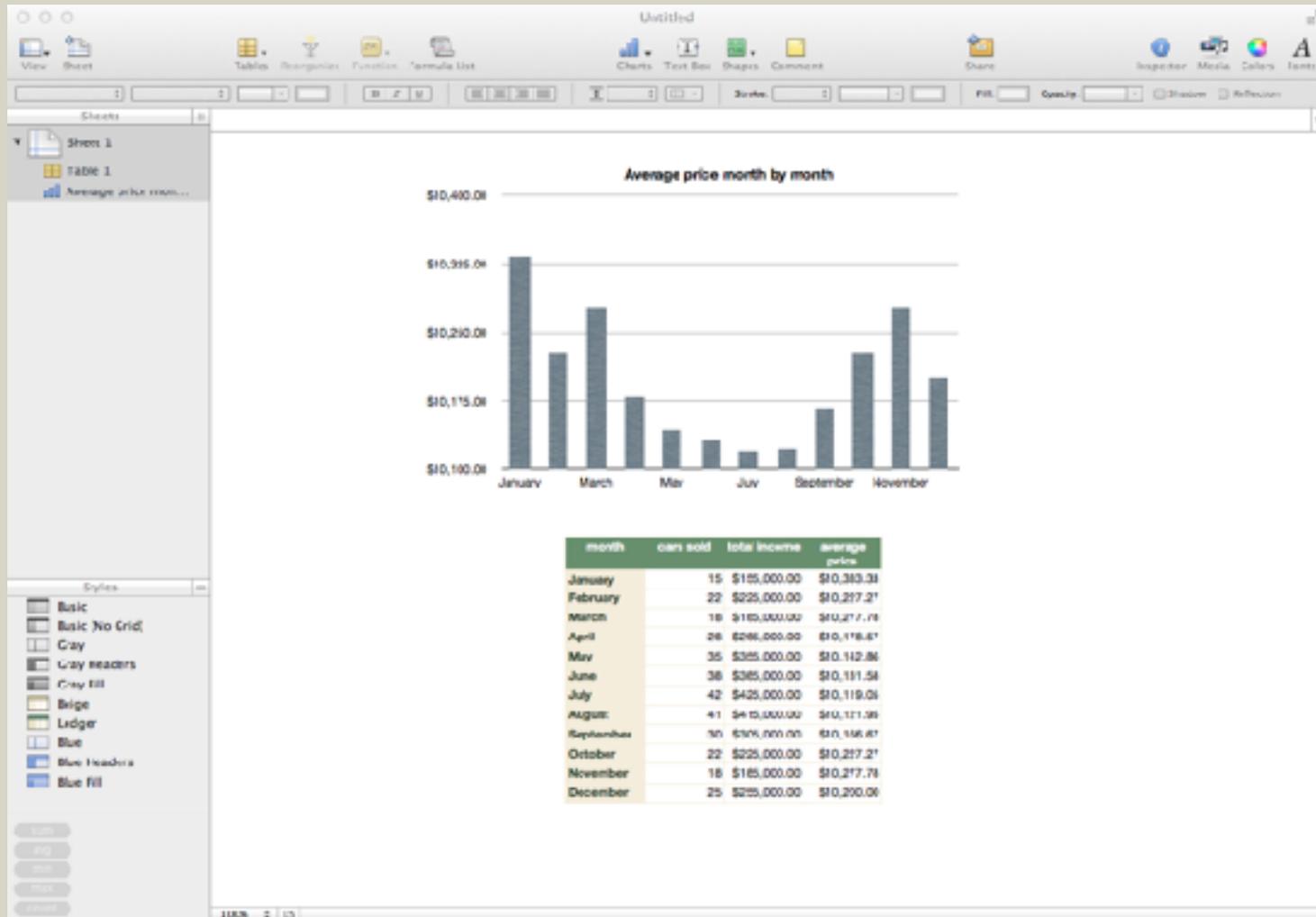
- Only first order functions are supported
 - As a result, functions like sumif, countif ... resort to hacks
 - What if I want an averageif? ...
 - oops, guess what, they added it!
 - which, of course, kind of proves the point!
- Functions are predefined
 - can create new ones only via VBA, XLL, ...
 - result: excessive reliance on copy and paste

P3:

model – view entanglement

- Worksheets have dual personality
 - They are the single data modeling construct ...
 - ... and the main mechanism to define named variables
 - Most users feel it's overkill to define a worksheet per variable
 - However: Worksheets \Leftrightarrow Tabbed Panels
 - ... and tabbed panels **are** the **primary visualization** mechanism!
- Organizing calculations is hard
 - Typical scenario: moving intermediate calculations out of the way
 - Best (modular) solution is to move them to separate worksheets
 - Sure, you can also hide rows and/or columns ...
 - ... but why should you work so hard to achieve this decluttering goal?
- Typical nightmare:
 - Rearranging the layout to make space for additional calculations
 - Making sure that different sheets share basic parameters
 - ... and visualizing said parameters in different sheets

Partial Exceptions: Numbers



Partial Exceptions: Javelin

The screenshot shows the Javelin spreadsheet interface. At the top, the word "income" is centered. Below it, a command line reads "income = < Monthly data: May 2011 - August 2011 >". A horizontal line separates this from the data table. The table has three columns of months and their corresponding income values. The first column contains data from May 2011 to Dec 2011. The second column contains data from Jan 2012 to Aug 2012. The third column contains data from Sep 2012 to Apr 2013. The "Sep 2011" row is highlighted in green. At the bottom of the window, the status bar shows "2047 276K", "NUM", "CALC", and "READY".

Month	Income	Month	Income	Month	Income
May 2011	500.00	Jan 2012		Sep 2012	
Jun 2011	650.00	Feb 2012		Oct 2012	
Jul 2011	720.00	Mar 2012		Nov 2012	
Aug 2011	811.00	Apr 2012		Dec 2012	
Sep 2011		May 2012		Jan 2013	
Oct 2011		Jun 2012		Feb 2013	
Nov 2011		Jul 2012		Mar 2013	
Dec 2011		Aug 2012		Apr 2013	

- Arguably Javelin could have set the standard for the future of spreadsheets, but failed for many reasons ... including bad luck!
- “A complex model can be dissected and understood by others who had no role in its creation, and this remains unique even today.” [Wikipedia]

Partial Exceptions: Lotus Improv

The screenshot displays the Lotus Improv interface with several overlapping windows:

- Menu:** Located on the left, it includes options like Info, File, Edit, Format, Worksheets, View, Graph, Print, Services, Hide, and Out.
- Share - Worksheet - Unfiled:** A spreadsheet showing market share data for various products and channels across the years 1985, 1989, 1990, and 1991. The data is organized by region (Galaxy, Snackers, Mintz, Paydit) and then by product and channel.
- Worksheet - Unfiled:** A second spreadsheet showing data for the years 1988, 1989, 1990, and 1991, with rows for different products and channels.
- Channel Share - Worksheet:** A smaller spreadsheet at the bottom left showing channel share data.
- Formula Bar:** At the bottom, it shows a formula: `=sum(Galaxy... Paydit) GIP All channels All products, Vending All regions 1988 All products... Supermarket All region 1991 All products`.

Region	Product	Channel	1985	1989	1990	1991
Galaxy	Vending	10.31%	8.58%	8.72%	8.84%	
		4.18%	3.94%	4.36%	4.34%	
		0.30%	15.78%	5.87%	15.55%	
Snackers	All channels	14.16%	28.31%	28.04%	29.53%	
		10.31%	8.58%	8.72%	8.84%	
		3.16%	3.45%	3.69%	3.68%	
Mintz	Supermarket	20.10%	18.47%	18.81%	14.87%	
		34.48%	28.54%	28.04%	27.59%	
		5.18%	6.96%	6.70%	6.47%	
Paydit	Grocery	3.10%	2.55%	2.57%	2.59%	
		16.10%	13.69%	13.41%	13.15%	
		28.18%	23.25%	22.88%	22.20%	
All products	Vending	5.72%	5.10%	5.36%	5.60%	
		3.15%	3.02%	3.35%	3.66%	
		14.31%	11.63%	11.62%	11.44%	
All channels	All channels	23.18%	19.95%	20.34%	20.89%	
		34.53%	29.23%	29.50%	29.14%	
		14.16%	12.99%	13.97%	14.87%	
All products	Supermarket	5.150%	57.77%	56.54%	55.19%	
		100.00%	100.00%	100.00%	100.00%	

Problems: summary

- Deficient data modeling constructs
 - All models must be represented as a set of 2D grids ...
 - ... with variables squeezed in a sea of (usually nameless) cells
 - All variables (cells) are of variant type
- Lack of functional abstraction
 - Can't define new functions to abstract repetitive formulas
 - ... except by going outside of the pure spreadsheet realm
 - No higher order functions
- Entanglement of model and presentation
 - Sharing parameters across worksheets is error prone
 - Rearranging sheet layouts is a frequent cause of bugs
 - Organizing calculations is unnecessarily hard
- Other issues
 - Lack of support for stateful computations
 - Wait ... What? This is **Lambda Daλs** ... This is **Functional Programming Sparta!**

Problems => Reactions => Symptoms

- Coping mechanisms:
 - copy-and-paste / cut-and-paste
 - naming of cells and ranges
 - multiple worksheets
 - Macros and VBA
- Consequences:
 - Spreadsheets are error prone
 - Hard to inspect and validate
 - Easy to break
 - Not that flexible
 - Not powerful enough for serious computing
 - ... see previous laundry list of issues

Smart Sheets Manifesto

- Spreadsheets must
 - Support richer data structures
 - orthogonal data composition
 - product types, sum types, ...
 - static data typing (possibly with “optionality” via variant)
 - Provide unfettered functional abstraction
 - functional composition: ability to define new functions combining existing ones
 - higher order functions: functions can be passed as arguments
 - Separate model from presentation
 - allows for multiple views of the computation model
 - Allow model sharing
- Let's aim higher
 - Why try to be “just like Excel” ...
 - ... when we can be better than Excel?

A Natural Approach: Build a Smart Sheet env over a good language

- Provide a visual interactive environment for a functional language, like Haskell
 - A lazy functional programming language works best
 - But an eager language works too ... with some tricks
- Definitely, the most elegant, sound, ... etc. approach
- But what about widespread **adoption**?
 - Spreadsheets do serve a purpose for non-programmers
 - “I can’t Haskell today ... I haz the dumbs”
 - The creator of Elm appears to think along similar lines
 - “Let’s be mainstream!”
 - “One factor is that we make things artificially hard to learn, sometimes with a seemingly **pathological glee**.”
 - He is not the only one who says things like this ...

ZenSheet

- A different approach:
 - Push the envelope generalizing spreadsheets
 - Exploit unifying principles
 - Define transformations from the “old way” to the “new way”
- **Downside:** with this approach inevitably leads to a programming language and environment that exhibits “the good parts” and “the bad parts”, however ...
- **Upside:** (operating hypothesis)
 - Result amenable to a wide spectrum of users
 - Allows for a smooth learning curve:
 - Beginners => Intermediate => Advanced
 - Assisted automated conversion from spreadsheets to smart sheets

ZenSheet

- Technical challenges:
 - Worksheet context:
 - $W!C8 :- A8 * B8;$
 - $W!F9 :- Conv!$A$4 * A9;$
 - i.e.:
 - $W[4, 8] :- W[1, 8] * W[2, 8];$
 - $W[6, 9] :- Conv[1,4] * W[1, 9];$
 - Which (clearly) are both horrible ways to express:
 - $position[8].value :- position[8].price * position [8].qty;$
 - $vehicle[9].kph :- conv.m2k * vehicle[9].mph;$
 - Or, better:
 - $position.value :- position.price * position.qty;$
 - $vehicle.kph :- conv.m2k * vehicle.mph;$

ZenSheet's XXI Century Type System

- Basic Types
 - boolean, integer, floating point, string
- N-Dimensional Arrays
- Product types
 - with unnamed (tuples) and named (structs) components
- Sum types (TBI)
 - Pattern matching rules
- Functions as first class citizens
- Gradual typing
- Lazy variables
- ... and a bit more

Type System Rules (Theorems)

two examples:

- Let
 - $\text{type}(A) = \text{array}[,]T$
 - $\text{type}(i) = \text{int}$
 - $\text{type}(j) = \text{int}$
- And
 - $0 \leq i < \text{dim}(A)[0]$
 - $0 \leq j < \text{dim}(A)[1]$
- Then
 - $\text{type}(A[i,j]) = T$

- Let
 - $\text{type}(p) = \text{struct} \{T0;T1;\}$
- And
 - $0 \leq i < 2$
- Then
 - $\text{type}(p[0]) = T0$
 - $\text{type}(p[1]) = T1$
 - $\text{type}(p[i]) = \text{var}$

Unifying result: “A worksheet is a 2-dimensional array of lazy variants”

Functional Abstraction

- It should be possible to
 - Assign functions to variables
 - Pass functions as arguments to other functions
 - Return functions as results
- Commonly used higher order functions (built-in)
 - filter
 - fmap
 - fold

Model – View Separation

... enough! ...

this is better illustrated via show and tell

It's light demo time!

Warnings:

this is a work in progress

crashes may occur

The syntax is tentative

Some semantic aspects are still experimental

Suggestions are welcome: really!

ZenSheet Studio Demonstration

Conclusions

- A computation model must stand on its own, entirely separated from its presentation
- An orthogonal and robust type system promotes understanding and increases users' ability to maintain correctness
- Functional abstraction is a must
 - And not just because this is **Lambda Daλs!**
- Presentation is a pillar of usability
 - The visual, intuitive, ease of use of spreadsheets should not, and need not, be lost as a consequence of supporting functional abstraction and a richer type system.
 - The pre-paid learning cost need not be thrown out
- Let's make functional programming popular by making it more accessible

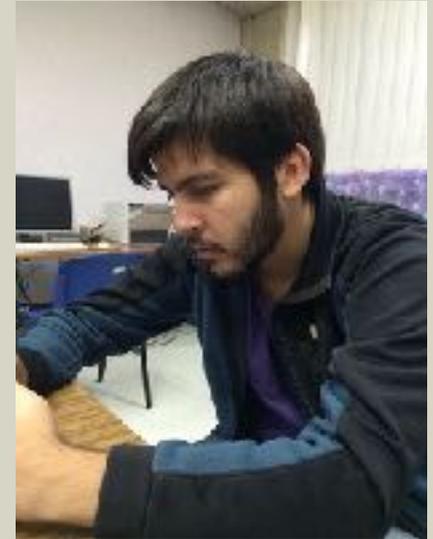
The ZenSheet Team



UNIVERSIDAD SIMÓN BOLÍVAR

By Universidad Simón Bolívar - <http://usb.ve>, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=24245670>

Lakebolt Research



Questions ... maybe answers too



By Kamizer - Own work, CC0, <https://commons.wikimedia.org/w/index.php?curid=18437040>