Inventing the "Treebook": A Workbook with Pages Linked in a Tree

Dave Sullivan[1]
College of Business, Oregon State University
Corvallis, Oregon 97330 USA

and

Matthew Garth McLuckie [2]
Group, WellMed, 520 NW Davis, Suite 300
Portland, Oregon 97209 USA

Abstract

A spreadsheet program is an ideal tool for recording scores and calculating grades—tasks every instructor needs to do. But anyone who has built a multipage workbook soon learns how difficult it can be to maintain formulas and entries among all pages. To help instructors sidestep these difficulties, we invented a "Treebook"; that is, an intelligent workbook whose pages are linked in a hierarchical tree. This article describes how we used Visual Basic to create two generations of Excel-based gradebooks that culminated in inventing the Treebook. We expect Treebooks will eventually be used in many application areas because they make building large spreadsheet models easier and more reliable.

Keywords: spreadsheet models, gradebook, Visual Basic, Treebook

1. PROBLEM STATEMENT

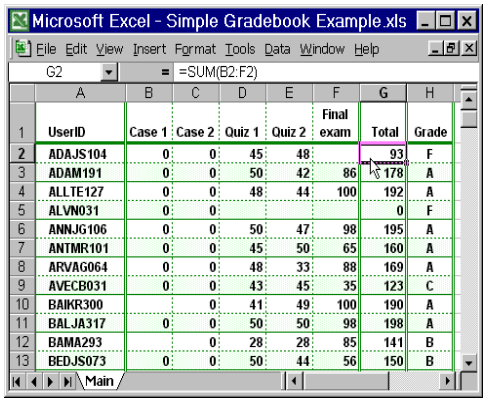
Instructors face a common set of housekeeping chores: maintain a class roster, grade activities, record scores, convert scores into an overall grade, and communicate evaluations to students. A spreadsheet program, such as Microsoft Excel, Corel Quattro Pro, or Lotus 1-2-3, provides an ideal tool for performing these tasks. Anyone comfortable writing spreadsheet formulas can create a simple gradebook solution in an hour or so.

Figure 1 shows the structure of a typical single-page gradebook application. Adding a new student or activity to this gradebook is relatively straightforward. To add a new student, a new row needs to be inserted, and formulas for the Total and Grade columns need to be copied into the new row. To add a new activity, a new column needs to be inserted, and formulas in the Total column may need to be modified if they no longer add activity scores correctly. Both operations have the potential to disrupt data in other areas of the page, such as a grade lookup table that might be stored below or to the right of the main gradebook area.

Dan Bricklin is popularly credited with inventing the spreadsheet in 1978 while a student at the Harvard Business School (Bricklin, 2000). Earlier programs had offered the ability to build financial models in a row-and-column format, but Bricklin's VisiCalc offered an interactive way to change values, scroll, and copy formulas with both relative and absolute addresses. Spreadsheet programs today retain many features pioneered in VisiCalc's original design, and with minor modifications, the gradebook shown in Figure 1 could have been built with VisiCalc.

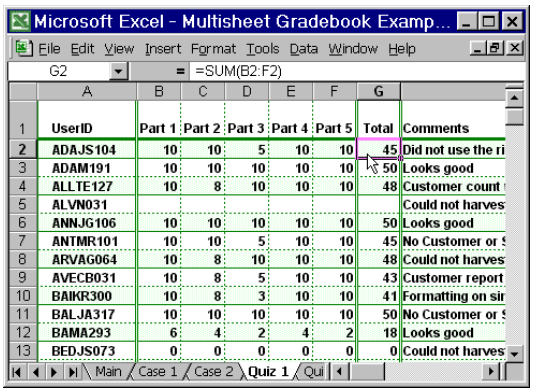
VisiCalc worksheets had only one page per spreadsheet file. So if the user wanted to store several different types

[1] Sullivan@bus.orst.edu
[2] MatthewMcLuckie@aol.com



	A	B	C	D	E	F	G	H
1	UserID	Case 1	Case 2	Quiz 1	Quiz 2	Final exam	Total	Grade
2	ADAJS104	0	0	45	48		93	F
3	ADAM191	0	0	50	42	86	178	A
4	ALLTE127	0	0	48	44	100	192	A
5	ALVN031	0	0				0	F
6	ANNJG106	0	0	50	47	98	195	A
7	ANTMR101	0	0	45	50	65	160	A
8	ARVAG064	0	0	48	33	88	169	A
9	AVECB031	0	0	43	45	35	123	C
10	BAIKR300		0	41	49	100	190	A
11	BALJA317	0	0	50	50	98	198	A
12	BAMA293		0	28	28	85	141	B
13	BEDJS073	0	0	50	44	56	150	B

Figure 1 A single-page gradebook. In this gradebook, the rows store data about each student and the columns store data about graded activities. The Total column summarizes all activity scores, and the Grade column uses a Lookup function to convert total points into letter grades.



	A	B	C	D	E	F	G	
1	UserID	Part 1	Part 2	Part 3	Part 4	Part 5	Total	Comments
2	ADAJS104	10	10	5	10	10	45	Did not use the ri
3	ADAM191	10	10	10	10	10	50	Looks good
4	ALLTE127	10	8	10	10	10	48	Customer count
5	ALVN031							Could not harves
6	ANNJG106	10	10	10	10	10	50	Looks good
7	ANTMR101	10	10	5	10	10	45	No Customer or S
8	ARVAG064	10	8	10	10	10	48	Could not harves
9	AVECB031	10	8	5	10	10	43	Customer report
10	BAIKR300	10	8	3	10	10	41	Formatting on sir
11	BALJA317	10	10	10	10	10	50	No Customer or S
12	BAMA293	6	4	2	4	2	18	Looks good
13	BEDJS073	0	0	0	0	0	0	Could not harves

Figure 2 This workbook uses different pages for each classroom activity. For example, the Quiz 1 page contains scores for one activity, and scores from its Total column are posted to the Quiz 1 column in the Main page.

of information in the same spreadsheet model, such as a grade lookup table and an area filled with activity scores, then each part needed to be in different areas of the page. This approach made jumping from one part of the model to another difficult. It also made common operations like inserting rows or columns dangerous because revisions to one area could inadvertantly damage other areas.

These problems were solved when a software development team at Borland extended Bricklin's idea to encompass a workbook full of worksheets with page tabs on the bottom. Borland filed a basic patent in April 1992 to cover these inventions (Anderson, 1992). The workbook makes larger models feasible; for example, in a gradebook, each graded activity can have its own page to store detailed comments or partial-activity scores. Other pages can summarize activities to arrive at an overall course grade or hold grade lookup tables, graphs, or other data. Jumping from one page to another involves clicking on page tabs. New rows or columns can be added to one page without affecting other pages adversely.

The workbook led to another surge in spreadsheet popularity because it let people build much larger, well-organized models. Workbooks let ordinary people tackle tasks that previously would have required setting up a database or writing programming code. Best of all, workbooks let people tap into a host of handy graphing and analysis features and offer a familiar geographical approach to exploring data.

Computer scientists cringe when they think about large spreadsheet models because workbooks do not enforce any sort of data integrity. Maintaining large workbook models quickly becomes a nightmare. Consider Figure 2, a typical gradebook application in a class with activities like case studies, quizzes, and a final exam. If information about each graded activity occupies a different page, then adding a new student to the workbook requires lots of steps:

Each page in the book needs a new row to hold the new student's scores, total, and grade.

1) The student's UserID must be added to each page—causing data to be stored redundantly.
2) Formulas need to be filled into the new rows so total columns will aggregate; grade columns will use a lookup table; and so on.
3) Other formulas need to be created to post results from one page to another.

Entirely different sequences of steps must be followed to add or delete new activities to the workbook or to sort items without jumbling data among pages. All these maintenance steps offer the potential for making hidden errors or introducing data inconsistencies.

We spent the last two years inventing ways to sidestep these problems. Our basic approach has been to make the workbook smarter about the objects it contains. That way, when an object like a student or activity is modified, the user needs to give only one command yet the workbook can be intelligent enough to modify all affected pages appropriately.

During the last two years, our ideas followed an evolutionary process, and we built two successive generations of intelligent workbooks. The first generation used Excel as a front-end for displaying information while all data were stored in a back-end Microsoft Access database. For the second generation, we constructed an Excel Add-in that lets a workbook contain a Treebook; that is, a collection of pages linked in a hierarchical tree. This second-generation approach makes the workbook smarter without needing to store data in a separate database. The rest of this article

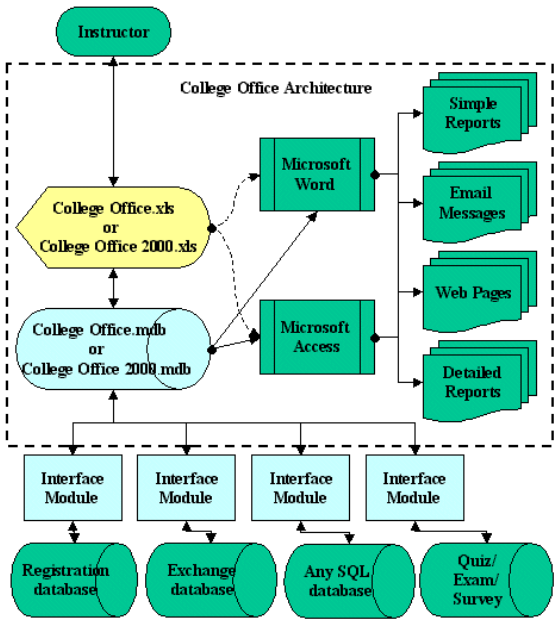


Figure 3 A flowchart showing the architecture used in the Suvan College Office gradebook.

describes each generation in more detail and explains how our thinking evolved.

2. FIRST GENERATION: SUVAN COLLEGE OFFICE GRADEBOOK

When we began building our first-generation gradebook, our design goal was simple: we wanted to combine the specialized features found in commercial standalone gradebook programs with the familiarity and ease-of-use of Excel.

To meet this goal, we planned to store all data—activity scores, spreadsheet formulas, student names, and so on—in an Access database. We intended to use Excel only as a front-end that displayed information and computed formulas but contained no permanently stored user-supplied data (see Figure 3). Finally, we decided to write all programming instructions in Visual Basic for Applications and store the resulting code inside an Excel workbook. We called our prototype "College Office" because it was built on top of Microsoft Office and was designed to help college instructors (Sullivan, 1999).

To the best of our knowledge, no one had tried this strategy of using a read-only Excel workbook as a front-end to an Access database before, and we rapidly began to understand why. Our first challenge came from slow performance. Every time a cell's value was changed in Excel, our code needed to trap the event, determine what part of the gradebook model had changed, post changes to tables in the Access database, and refresh the Excel display appropriately. Since Visual Basic for Applications is interpreted rather than compiled, the computer had to interpret each statement as well as execute its instructions. Much larger delays came from passing information between Excel and Access. Microsoft calls these data flows "out of process" communication, and they seemed to take forever to execute. Similar problems occurred whenever our code needed to drop data on a worksheet. We nearly abandoned the project after the first month because it looked like the performance would remain too sluggish for the program to work reasonably. However, with experimentation, we discovered ways to pack all inter-process communications into arrays. Since the communication overhead was based on how many calls were made between programs rather than how much data were exchanged, using arrays judiciously caused the final version of College Office to run at least one hundred times faster than our initial prototypes.

We soon ran into another problem. We wanted to allow instructors to give a single command to add or delete an activity and have the program make all necessary modifications to the workbook. These modifications required adding or deleting columns from the top-level page, creating or deleting the associated second-level activity page, adjusting formulas, and so on. The problem came from Microsoft Excel 97's unreliability at creating new pages via Visual Basic commands. Most of the time it would create new pages without a hitch, but occasionally our code would freeze the computer when Excel tried to create a new page. No amount of testing or debugging could solve this problem, so we stopped asking Excel to create new pages programmatically. Instead, we built a workbook with 50 hidden pages, and when the user gives a command requiring a new page, the code unhides an existing page. While this approach worked, it made the workbook large because each page adds 50K to the workbook's size.

It took us a year of full-time effort to design, build, test, document, and release the Suvan College Office gradebook program. After testing the program ourselves, we conducted beta-level testing at Oregon State University and Linn Benton Community College. As soon as we finished testing, Microsoft included the program in its next Online Learning Resource Kit (Suvan LLC, 1999). Our software required its own CD-ROM in this kit because we included forty-two videos showing how to set up and use the gradebook. We are grateful to Microsoft for the support they provided our project, both for the license fees they paid Suvan LLC and for shipping 40,000 copies of the program to instructors worldwide as a public service.

Software is dynamic—full of motion and visual activity—so this article cannot capture the richness of how Suvan College Office operates. If you want to learn

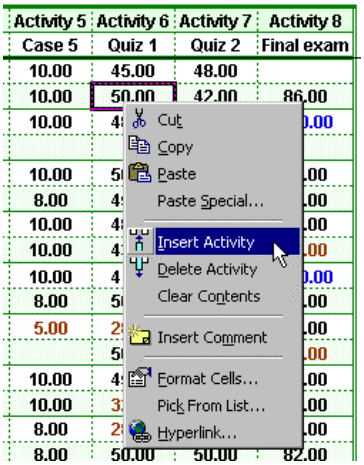


Figure 4 Clicking with the right mouse button causes this pop-up menu to appear in College Office. Choosing the Insert Activity command will insert a new activity column at the current location and add a new activity page to the gradebook. For example, this Insert Activity command will add a new Activity 6 and renumber the current Activity 6 to become Activity 7.

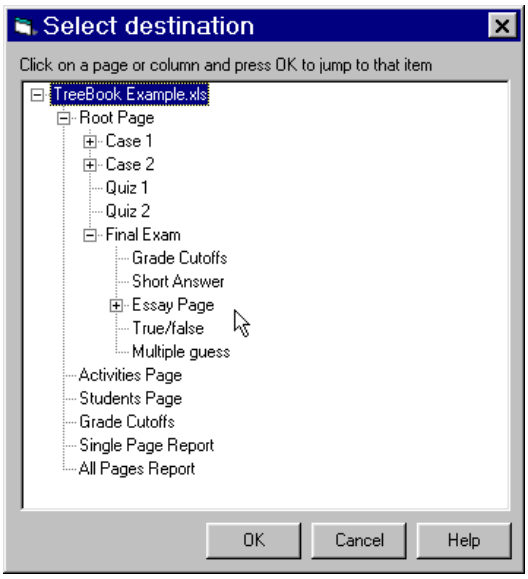


Figure 5 The Treebook Explorer view shows how Treebook pages are linked in a hierarchy.

more about the program, the best strategy would be to obtain a copy and try using it. Microsoft's Online Learning Resource Kit is now out of print, but you can download the program for free from Microsoft's Office Update website (Microsoft, 1999) or Suvan LLC's website at www.collegeoffice.com.

If you know something about Visual Basic programming, then you can look at the source code. The program uses 13,000 lines of carefully documented code. The code is password protected, but the password is "treasure". Use all lowercase characters for the password, and do not type the quote marks.

3. SECOND GENERATION: SUVAN TREEBOOK

Just after we released our Suvan College Office gradebook program, Microsoft released its Office 2000 Developers Kit. We soon discovered the limitations and obstacles that had plagued us throughout the last development cycle had disappeared. The Office 2000 development tools let us remove all code from the Excel workbook and compile it as a separate Excel Add-in. This would let us avoid warnings about embedded macros whenever a workbook is opened. With full access to all tools in the Visual Basic 6 Enterprise Edition, we had improved testing and debugging aids and better support for building large object models. Over time we discovered the development tools' reliability had gotten better as well. All this prompted us to start from scratch and develop an entirely new way to make Excel workbooks more intelligent.

A key insight came when we considered how to organize pages in the workbook. Suvan College Office had used a main page to summarize information coming from child pages, but it didn't let child pages have grandchild pages. We decided to remove this limitation and support a linked set of hierarchical pages where each page could post data to a parent page and receive data from child pages. As an example of how this works, examine the page structure shown in Figure 5: the Root page summarizes data from the Case 1, Case 2, and Final Exam pages as well having Quiz 1 and Quiz 2 columns. At the next level down in the hierarchy, the Final Exam page summarizes data from an Essay page and from the Grade Cutoffs, Short Answer, True/False, and Multiple Choice columns.

Figure 6 shows an even smaller Treebook with only three pages. It shows how values on two child pages get posted to a parent page.

A Treebook can extend or collapse dynamically depending on how much detail the user wants to store or analyze. Not only is this helpful to the user, but the treelike structure made writing the Visual Basic code easier because all pages operate the same regardless of their position in the hierarchy.

The sample Treebook in Figure 7 uses typography to indicate which columns have associated child pages and which do not. A cell whose content is computed by a formula appears in bold face; a cell that contains a value

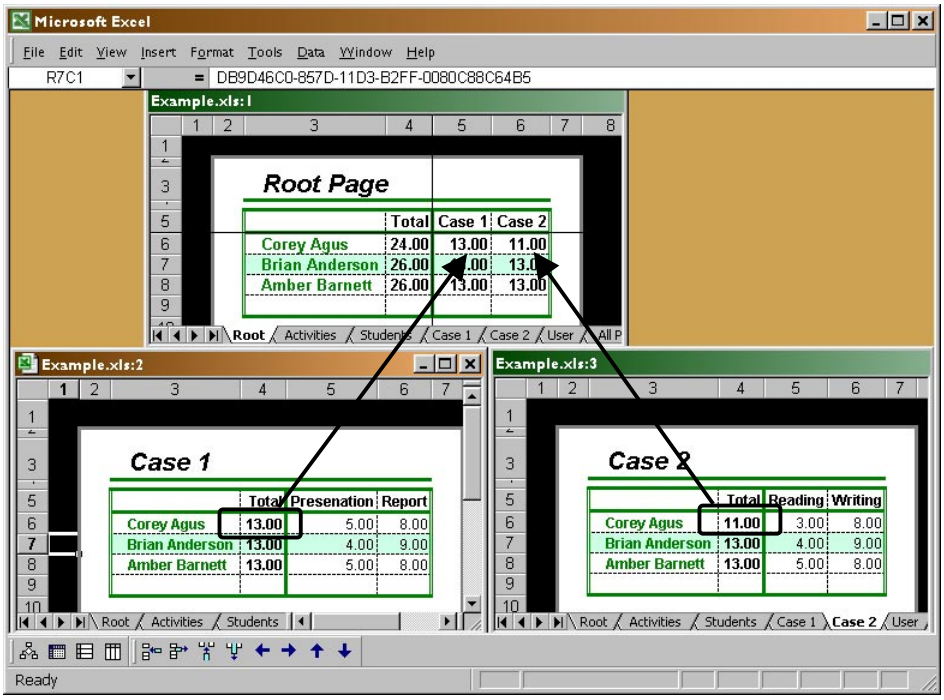


Figure 6 These three pages show how values on two child pages get posted to a parent page.

appears in regular face. Because the cells in the Quiz 1 and Quiz 2 columns appear in regular face, they contain values and do not use formulas to obtain their values from a child page.

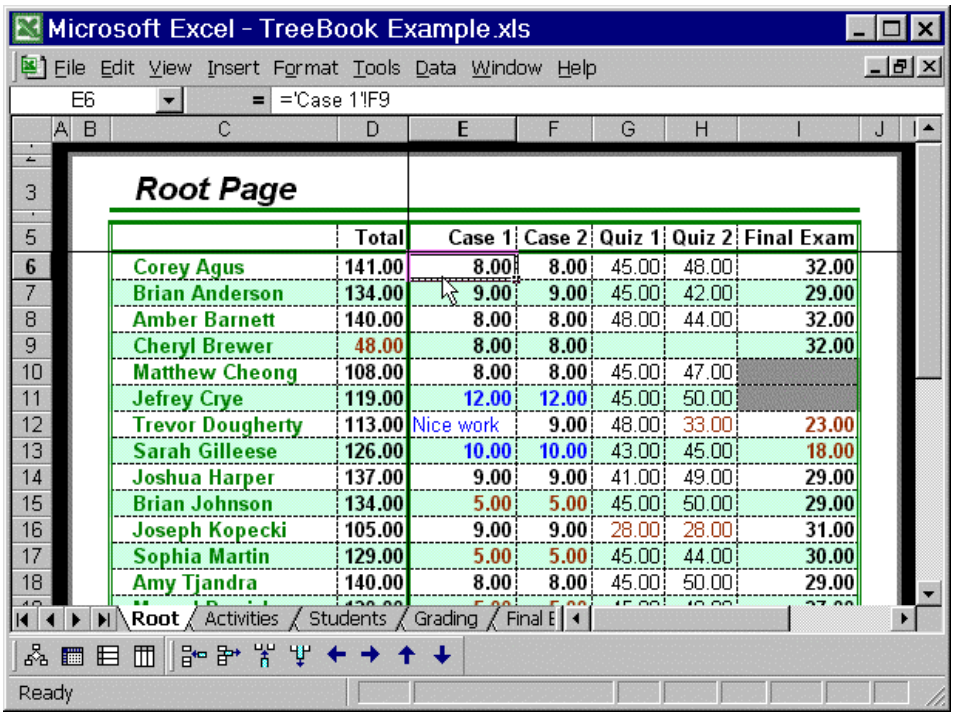
Suppose the user decides to record detailed information about one of these quizzes. The easiest way to request a child page is to double-click in a column. Thus, by double-clicking on the Quiz 1 column, the user would be asking the Treebook Add-in to create a new page named Quiz 1, copy the Quiz 1 column's values from the current page down to the newly constructed child page, and place formulas in the Quiz 1 column that obtain values from the child page's Total column.

This example raises an important question: Where does a Treebook store information about conceptual objects like students and activities? Suvan College Office stores this information in a back-end Access database, and we decided not to use this approach again. Storing and retrieving data from Access is slow. Even worse, many of our user support questions came from difficulties with installing and referencing the database engine—something largely outside our control.

We decided to store this sort of information in hidden areas of the workbook itself. We placed information about the Treebook's hierarchical organization in a Hidden Structure page; information about each page's contents was hidden in the page's top two lines and first two columns. So whenever a workbook is opened, our Treebook Add-in looks for a Hidden Structure page. Workbooks without a Hidden Structure page are ordinary workbooks and are ignored by our Treebook Add-in. If the Treebook Add-in finds a Hidden Structure page, it reads the page to determine which other pages are part of the Treebook, how many rows and columns each Treebook page has, and so on. The typical user never sees any of this hidden structural data.

All these data are loaded into various Visual Basic collections in memory that represent the objects in the Treebook. For example, data from the Hidden Structure page determines how many instances of the Treepages collection (the Treebook equivalent of a worksheet) should be constructed. Other important collections include Rowcols (the Treebook equivalent of rows or columns within a spreadsheet model), Pagerowcols (an instance of a Rowcol on a specific Treepage) and Cells. Our Treebook Add-in loads the entire Treebook into these collections, but future Treebook Add-in implementations could open faster by loading only part of the Treebook into memory.

Treebook objects are smarter than their spreadsheet equivalents. For example, a Treepage knows:



	Total	Case 1	Case 2	Quiz 1	Quiz 2	Final Exam
Corey Agus	141.00	8.00	8.00	45.00	48.00	32.00
Brian Anderson	134.00	9.00	9.00	45.00	42.00	29.00
Amber Barnett	140.00	8.00	8.00	48.00	44.00	32.00
Cheryl Brewer	48.00	8.00	8.00			32.00
Matthew Cheong	108.00	8.00	8.00	45.00	47.00	
Jefrey Crye	119.00	12.00	12.00	45.00	50.00	
Trevor Dougherty	113.00	Nice work	9.00	48.00	33.00	23.00
Sarah Gilleese	126.00	10.00	10.00	43.00	45.00	18.00
Joshua Harper	137.00	9.00	9.00	41.00	49.00	29.00
Brian Johnson	134.00	5.00	5.00	45.00	50.00	29.00
Joseph Kopecki	105.00	9.00	9.00	28.00	28.00	31.00
Sophia Martin	129.00	5.00	5.00	45.00	44.00	30.00
Amy Tjandra	140.00	8.00	8.00	45.00	50.00	29.00

- Where it fits in the Treebook's hierarchy of pages.
- Which Rowcols appear in its PageRowCols collections.
- A RowCol knows:
- Its key, which uniquely identifies a logical row or column. A logical row or column might appear on more than one Treepage. This would happen when a new student is added to a Treebook, and the student's Rowcol appears on several Treepages.
- Whether the Rowcol is part of the Rows or Columns collection. Since any spreadsheet model can be transposed to make rows into columns and visa versa, our Visual Basic code treats both rows and columns essentially the same by creating them from the same Rowcol class.

A Cell knows:

- Its key, which is built from its row's Rowcol key and its column's Rowcol key. Thus, a cell's key uniquely identifies the cell within the Cells collection, and it also identifies where the cell falls in the various Rowcol and Pagerowcol collections.
- Whether it stores a value or a reference to another cell. An important design feature of a Treebook is to store each logical piece of data in only one cell. Thus, although a student's name might appear on many pages in a Treebook, only one cell will store the name, and all other cells will reference that cell. The user doesn't need to know which cell is the owner and which cells reference the owner. If the user enters a value in a cell that stores a reference to another cell, the Treebook Add-In will place the value in the correct cell and replace the reference in the changed cell.

The net effect of all these collections is to teach the Treebook about the objects it contains, such as students, courses, and classroom activities. This lets the user concentrate on building a spreadsheet model composed of logical objects that can behave in more intelligent ways than ordinary spreadsheet rows, columns, and cells.

So far all our examples have come from one area: building a gradebook. We hope Treebooks will find many other application areas. To help make this a reality, we built a design mode into our Treebook Add-in that reveals the contents of the Hidden Structure page. This lets someone without programming experience change what the Treebook objects are called. For example, to build a budgeting application, an accountant might change "Students" and "Activities" to be "Expenses" and "Time Periods". This sort of application could help people consolidate budgets throughout an

enterprise. A scientific application might analyze "Subjects" and "Experiments". Marketing professionals might analyze "Products" and "Sales Areas".

The static descriptions in this article do not do justice to the dynamic and interactive nature of working with a Treebook. We have finished writing a functional Treebook Add-in, but we have not finished alpha-level testing. So as we write this article, few people have seen the Treebook Add-in in action, and we have not released the program or its source code for general use. We will have completed internal testing by the time this paper is presented or published. At that time you should be able to obtain a copy for free from our Treebook website at http://www.Treebook.com.

4. SUMMARY AND CONCLUSIONS

This article described two ways to build more intelligent spreadsheet-based workbooks. The Suvan College Office gradebook program uses the first method, and this article describes how you can download and examine its source code. Instructors around the world are using this program to manage their classroom record-keeping chores.

The second method uses an Excel-based Add-in to hide structural information inside a workbook, thereby converting it into a Treebook. A Treebook makes building and maintaining data-intensive models much easier than trying to link worksheets by hand.

Just as with a standard workbook, a Treebook can build just about any sort of model. The main difference comes because a Treebook knows about the items that it contains, so it can maintain relationships among the items in a more intelligent manner.

Although our Treebook Add-in uses specific techniques to work its magic, a Treebook's features could be added to nearly any spreadsheet program or built with other programming languages.

Our long-run hope is that people will adapt the Treebook concept to serve as a front-end to a relational database or OLAP server. This would let people suck large amounts of data into a well-structured spreadsheet model. At least in theory, this could combine the geographic familiarity and analytical power of a spreadsheet with the organization and integrity of a database.

5. REFERENCES

Anderson, Charles; et al. (1992, April 8). *US Patent #5,416,895: System and methods for improved spreadsheet interface with user-familiar objects.* Retrieved March 5, 2000 from the IBM Intellectual Property Network on the World Wide Web: http://www.patents.ibm.com/

Bricklin, Dan (2000). *VisiCalc: Information from its creators,* Dan Bricklin and Bob Frankston. Retrieved May 28, 2000 from the World Wide Web: http://www.bricklin.com/visicalc.htm

Microsoft Corporation. (1999, November 8). *Suvan Office Gradebook for Excel 2000 and Access 2000.* Retrieved May 29, 2000 from Microsoft Office Update on the World Wide Web: http://officeupdate.microsoft.com/

Sullivan, Dave. (1999, June). *College Office User Manual*. Retrieved May 14, 2000 from the World Wide Web: http://www.collegeoffice.com/manual.doc

Suvan LLC (1999, Spring). Suvan College Office. *In Online Learning Resource Kit* (Vol. 2, CD-ROM #3). Kent, WA: Microsoft Corporation