Applying a "What You See Is What You Test" (WYSIWYT) Technology to Commercial Spreadsheet Packages: Several Scenarios

Margaret Burnett and Gregg Rothermel
Oregon State University

Introduction

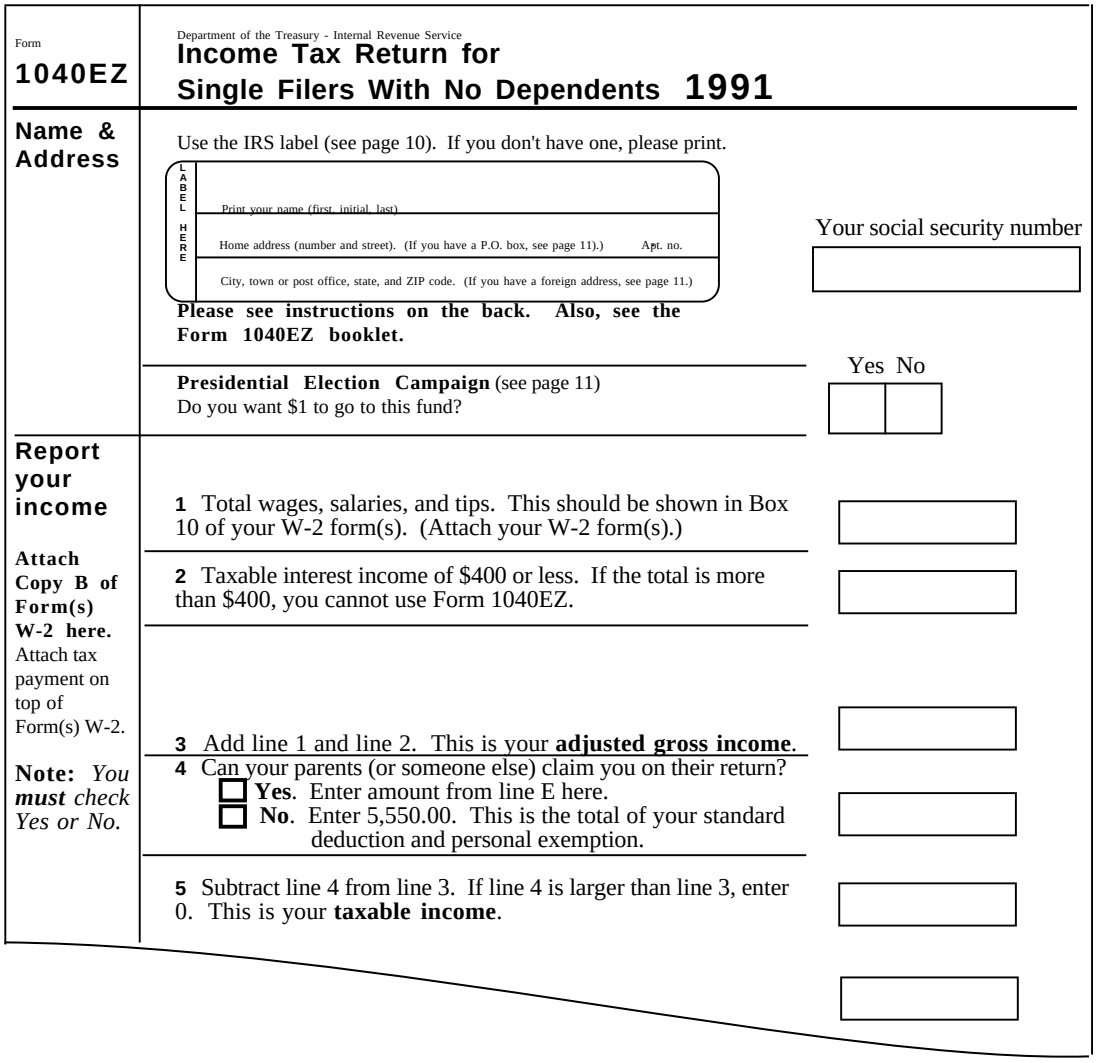
The "What You See Is What You Test" (WYSIWYT) methodology is a new approach to software engineering in highly visual problem-solving environments such as spreadsheets. The methodology is designed to support end users as well as more sophisticated developers. It integrates the knowledge of testing and debugging procedures into such an environment, to allow a user to become more effective at testing and debugging, even without prior training in software engineering principles.

The WYSIWYT methodology has already been integrated into the research spreadsheet language Forms/3. Here are a few scenarios illustrating how it might look if integrated into commercial spreadsheet packages.

Scenario 1: An end user figures out and tests her income taxes

An end user has a printout of a tax form from the IRS such as in Figure 1 in front of her, and she wants to use a spreadsheet to figure out the answers. To do this, she has created a spreadsheet such as the one in Figure 2.

Although this spreadsheet is simple, there are several ways the user could end up reporting the wrong answer. Like many taxpayers, she may be struggling to gather up all the required data, and may change her mind about the right data values to enter. If she has been taking shortcuts with the formulas, basing them upon the conditions present in her first version of the data (such as not bothering to use a *max* operator in line 5 to prevent negatives), the formulas are probably not very general, and may cause problems if her data changes. For example, if she entered "line 4 - line 3" as the formula for line 5, but later changes line 4 to 5500 because her parents tell her they did not claim her this year after all, then the formula for line 5 will not give the correct answer. Similar problems could arise if she discovers that she entered data from the wrong box of her W-2, and so on.



Form **1040EZ**

Department of the Treasury - Internal Revenue Service

Income Tax Return for Single Filers With No Dependents 1991

Name & Address

Use the IRS label (see page 10). If you don't have one, please print.

LABEL HERE

Print your name (first, initial, last)

Home address (number and street). (If you have a P.O. box, see page 11.) Apt. no.

City, town or post office, state, and ZIP code. (If you have a foreign address, see page 11.)

Your social security number

Please see instructions on the back. Also, see the Form 1040EZ booklet.

Yes No

Presidential Election Campaign (see page 11)
Do you want $1 to go to this fund?

Report your income

Attach Copy B of Form(s) W-2 here. Attach tax payment on top of Form(s) W-2.

Note: *You **must** check Yes or No.*

1 Total wages, salaries, and tips. This should be shown in Box 10 of your W-2 form(s). (Attach your W-2 form(s).)

2 Taxable interest income of $400 or less. If the total is more than $400, you cannot use Form 1040EZ.

3 Add line 1 and line 2. This is your **adjusted gross income**.

4 Can your parents (or someone else) claim you on their return?
☐ **Yes**. Enter amount from line E here.
☐ **No**. Enter 5,550.00. This is the total of your standard deduction and personal exemption.

5 Subtract line 4 from line 3. If line 4 is larger than line 3, enter 0. This is your **taxable income**.

Figure 1: A portion of a tax form from the IRS.

1040EZ calculations:				
Presidential election?	yes			
1. Total wages	$5,132			
2. Taxable interest	$297			
3. Adjusted gross	$5,429			
4. Parents?	$1,500		Line E	$1,500
5. Taxable income	$3,929			

Figure 2: The user's spreadsheet to figure out the taxes. The first few cells are simply data values. Line 3's formula is line 1 + line 2, line 4's formula is a reference to line E, and line 5's formula is line 3 - line 4.

Even in this simple case, the WYSIWYT methodology can help. Figure 3 shows a mock-up of how it might be incorporated into a popular spreadsheet package. All cells containing formulas (as opposed to data values) are initially red-bordered with checkboxes, as in Figure 3(a). The first time the user sees a red border, she moves her mouse over it and the tool tips inform her that "red borders mean untested and blue borders mean tested. You can check cells off when you approve of their values." The user checks off a value that she is sure is correct, and a checkmark (√) appears as in Figure 3(b). Further, the border of this explicitly-approved cell, as well as of

cells contributing to it, becomes blue. If she then changed some data, any affected checkmarks would be replaced with question marks (?). This would remind her to check again the cells whose values she thought were important enough to check off before. But instead of replacing a data value, suppose the user makes the formula change alluded to above in line 4, changing the previous formula to the constant 5500 instead of the former reference to line E. Since the change she made involved a formula (the one she just changed to a data value), the affected cells' borders revert to red and downstream √s and ?s disappear, indicating that these cells are now completely untested again. See Figure 3(c). The maintenance of the "testedness" status of each cell throughout the editing process, as illustrated in Figure 3(c), is an important benefit of the approach. Without this feature, the user may not realize that the testing she did before became irrelevant with her formula change and now needs to be redone.

A primary goal of this approach is to reduce overconfidence about the correctness of spreadsheet formulas. Our empirical work (e.g., [Rothermel et al. 1999]) shows that the methodology does significantly reduce overconfidence about how tested the spreadsheet is, as well as improving effectiveness and efficiency of testing and debugging.

1040EZ calculations:				
Presidential election?	yes			
1. Total wages	$5,132			
2. Taxable interest	$297			
3. Adjusted gross	☐ $5,429			
4. Parents?	☐ $1,500		Line E	$1,500
5. Taxable income	☐ $3,929			

(a)

1040EZ calculations:				
Presidential election?	yes			
1. Total wages	$5,132			
2. Taxable interest	$297			
3. Adjusted gross	☐ $5,429			
4. Parents?	☐ $1,500		Line E	$1,500
5. Taxable income	√ $3,929			

(b)

1040EZ calculations:				
Presidential election?	yes			
1. Total wages	$5,132			
2. Taxable interest	$297			
3. Adjusted gross	☐ $5,429			
4. Parents?	$5,500		Line E	$1,500
5. Taxable income	☐ -$71			

(c)

Figure 3: A mock-up of a popular spreadsheet package if enhanced by the WYSIWYT technology: (a): All cells containing formulas are initially red, meaning untested.
(b): Whenever the user makes a decision that some data value is correct, she checks it off. The checkmark appears in the cell she explicitly validated, and all the borders of cells contributing to that correct value become more tested (closer to pure blue). This example has such simple formulas, only the two colors red and blue are needed.
(c): The user changes the formula in line 4 to a constant. This change causes affected cells to be considered untested again.

Scenario 2: The user tests her income tax spreadsheet as she makes it more reusable

The next year, the user may want to improve the spreadsheet so that she can use it year after year without having to redesign each formula in the context of the current year's data values. For example, she adds the yes/no box from the IRS form's line 4 to her spreadsheet's line 4 and uses the *if* operator in the formula for line 4. Because of this *if*, she will need to try at least 2 test cases for line 4's cell to be considered tested: one that exercises the "*yes*" case and one that exercises the "*no*" case. (See [Rothermel et al. 1998] for a description of the coverage criteria currently in use as well as other possible criteria that can alternatively be employed.)

Because of this, when the user checks off one data value as in Figure 4, the border for lines 4 and 5 turn purple (50% blue and 50% red). To figure out how to make the purple cells turn blue, the user selects one of them and hits a "show details" key. The system then draws arrows pertaining to the subexpression relationships, with colors depicting which cases still need to be tested. The arrow from the last subexpression is red, telling the user that the "no" case still needs to be tried.

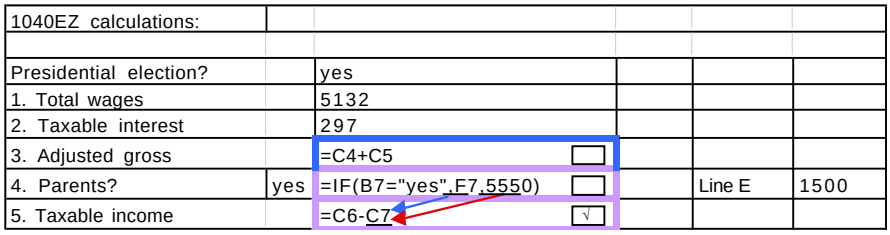


Figure 4: Some cells require more than one test value to become completely tested, as this formula view with purple cell borders and red and blue arrows between subexpressions shows.

Scenario 3: A template developer tests an income tax spreadsheet for sale

It is well documented that many production spreadsheets contain bugs. To help address this problem, a developer with a full suite of income tax spreadsheet templates for sale could use the methodology to achieve organized test coverage of these income tax spreadsheets. This would not only be valuable when first developing the spreadsheets, but also in making sure that each formula change in subsequent years' revisions had been entered and tested.

Replicated formulas

When the user has a collection of formulas that have been replicated, such as a teacher's spreadsheet of students' course grades in a course of 100 students, testing each cell with a formula replicated from another, previously-tested cell does not seem not productive. Instead, what is needed is a way to track testing of each *formula* rather than each *cell*. The WYSIWYT technology includes a way to do this that brings greater efficiency than cell-by-cell testing to both the user and the system, and yet is as visual and straightforward to use as are the above scenarios. See [Burnett et al. 1999] for details.

Debugging

We have expanded the WYSIWYT technology so that it also helps the user track down bugs by process of elimination. The user's input into this process is that, besides entering a √ for a value that is correct, she can also enter an X when she discovers a value that is incorrect. The system "subtracts" the set of correct cells from the set of incorrect cells in a manner similar to dicing to find and highlight the cells in which the bug could reside. See [Reichwein et al. 1999] for details.

Additional information

The WYSIWYT technology is being developed at Oregon State University by Margaret Burnett, Gregg Rothermel and several students, in collaboration with Curtis Cook and Thomas Green. For more information on this project, see http://www.cs.orst.edu/~grother/vptestdebug.html. Patents pending.

References

[Burnett et al. 1999] M. Burnett, A. Sheretov, and G. Rothermel, "Scaling Up a 'What You See is What You Test' Methodology to Spreadsheet Grids," *1999 IEEE Symposium on Visual Languages*, Tokyo, Japan, pp. 30-37, September 13-16, 1999.

[Reichwein et al. 1999] J. Reichwein, G. Rothermel, and M. Burnett, "Slicing Spreadsheets: An Integrated Methodology for Spreadsheet Testing and Debugging," *Conference on Domain-Specific Languages*, Austin, Texas, pp. 25-38, October 3-5, 1999.

[Rothermel et al. 1998] G. Rothermel, L. Li, C. DuPuis, and M. Burnett, "What You See is What You Test: A Methodology for Testing Form-Based Visual Programs," *International Conference on Software Engineering*, pp. 198-207, April 1998.

[Rothermel et al. 1999] K. Rothermel, C. Cook, M. Burnett, J. Schonfeld, T. Green, and G. Rothermel, "WYSIWYT Testing in the Spreadsheet Paradigm: An Empirical Evaluation," TR 99-60-08, Oregon State University, Computer Science Department, December 1999.