Overcoming Microsoft Excel's Weaknesses for Crop Model Building and Simulations

Christopher Teh Boon Sung*

article

ABSTRACT Using spreadsheets such as Microsoft Excel for building crop models and running simulations can be beneficial. Excel is easy to use, powerful, and versatile, and it requires the least proficiency in computer programming compared to other programming platforms. Excel, however, has several weaknesses: it does not directly support loops for iterative calculations, and it does not allow one cell to alter the contents of another cell. Thus, the objective of this study was to develop an Excel add-in, called BuildIt, that overcomes some of Excel's weaknesses by: (1) providing a loop for repetitive calculations and (2) providing several operations (called actions) typically needed in building crop models. These actions are such as for numerical integration, initialization of variables, and solving differential equations using the Runge-Kutta method, as well as for copying and manipulation of cell ranges. BuildIt was written in Excel's script language, Visual Basic for Applications (VBA), but it does not require users to program in VBA to build their models. Several examples of models were used in this article to illustrate how BuildIt implements the infrastructure in Excel, and how it can be used to build models and run model simulations. With BuildIt, users are able to use Excel to build and run their mathematical models, without requiring any knowledge in VBA.

Spreadsheets, in particular Microsoft Excel, are widely used in the world today. The immense popularity of spreadsheets is mainly due to their ease of use, power, and versatility. Spreadsheets provide many features such as: (1) numerical and non-numerical functions for mathematics, statistics, database, date and time, finance, logic, information, and engineering; (2) database representation and access; (3) charting and graphing; (4) display and document formatting capabilities such as layout, fonts, and colors to improve presentation; and (5) script or programming language such as VBA (Visual Basic for Applications) in Microsoft Excel and OOoBasic (OpenOffice.org Basic) in OpenOffice.org Calc. Originally conceived as a tool for financial analysis, spreadsheets have today evolved to become versatile tools to manipulate, analyze, present, and interpret data, as well as to build models for simulating various phenomena in science and engineering (Khandan, 2002).

There are many software tools to help in building mathematical models. These tools can be categorized into four groups: (1) general purpose computer languages (such as C, C++, Fortran, BASIC, and Java); (2) specialized simulation applications (such as PowerSim, Stella, and ExtendSim); (3) equation solver-based applications (such as Maple and Mathematica); and (4) spreadsheet-based applications (such as Microsoft Excel and OpenOffice.org Calc). One disadvantage of the first three groups is that they require users to have a high level of proficiency in computer programming. However, the fourth group, the spreadsheet applications, require the least level of proficiency in computer programming (Nardi and Miller, 1990; Brown, 1999; Khandan, 2002; Seila, 2005).

Spreadsheets relieve users from many concerns typically involved in computer programming such as maintaining the program flow and program design. The open tabular layout of spreadsheets provides users a simple, straightforward framework in which to build their models. Users need only to understand two basic concepts: to treat the cells as variables and the functions (or formulas) as the relationship between these variables (Nardi and Miller, 1990). Users specify the way variables depend on one another via formulas or functions, and the spreadsheet system maintains these variable dependencies. So, if one part of the spreadsheet changes, it triggers a calculation update to the whole spreadsheet so that all the dependent variables are automatically recalculated to reflect their new values. The order in which the variables are calculated is worked out by the spreadsheet

Department of Land Management, Faculty of Agriculture, Universiti Putra Malaysia, 43400 UPM Serdang, Selangor, Malaysia. Received 27 Aug. 2010. *Corresponding author (cbsteh@yahoo.com).

J. Nat. Resour. Life Sci. Educ. 40:122–136 (2011)
doi:10.4195/jnrlse.2010.0024 • http://www.JNRLSE.org

5585 Guilford Road, Madison, WI 53711 USA

Abbreviations: A_L, leaf photosynthesis; A_m, maximum leaf photosynthesis rate; AT, canopy photosynthesis; $A_{t,d}$, daily canopy photosynthesis; DGC, daily generic crop; DOY, day of year; e, solar radiation conversion factor; INI, BuildIt action for initialization; I_o, hourly total solar irradiance; $I_{t,d}$, daily total solar irradiance; ITG, BuildIt action for numerical integration; k, canopy extinction coefficient; k_d, degradation rate; k_g, growth rate; k_s, senescence rate; L, leaf area index; LAI, total leaf area index; RUN, BuildIt action for running a user-defined VBA script; SVP, saturated air vapor pressure; t_h, local solar hour; t_{sr}, time of sunrise; t_{ss}, time of sunset; UPD, BuildIt action for solving differential equations; VBA, Visual Basic for Applications; W_g, structural dry weight; W_s, storage dry weight; Y_g, efficiency of structural synthesis.

system based on the variable dependencies. Without automatic flow control, users will have to write programming code to track variable dependencies and to update all affected variables iteratively when required.

The open tabular layout of spreadsheets can help in model debugging or error-checking because the open layout reveals the values of all variables, including values of provisional and intermediary variables. When all values are visible, the presence of nonsensical, extreme, or dubious values is more easily detected by users (Brown, 1999). Simple sensitivity tests can also be carried out easily and quickly. Users can alter the value for one or more model parameters, and this alteration would trigger the spreadsheet's automatic update feature. Calculations are automatically performed again to update the values in all affected cells. In other words, users receive immediate feedback (such as the amount of change) when one or more model parameters are altered. Additionally, Excel provides the What-If Analysis for more advanced sensitivity tests. The strength of spreadsheets in data manipulation, charting, and formatting are particularly useful in presenting and analysing data. For instance, Excel's copious built-in functions and data utilities (such as Pivot Table and AutoFilter) can be used to manipulate the data to help in interpreting the model simulation results. Simulation results can also be formatted or charted in the spreadsheet itself without requiring users to export the results into a separate application for further analysis and interpretation.

Nonetheless, spreadsheets like Excel have several important drawbacks, limiting their use in mathematical modeling. Consequently, spreadsheets are sometimes merely used as a "scratch pad" or "test bed" to test modeling ideas or to test some parts of a model before the model is implemented entirely elsewhere. One key limitation of spreadsheets is they do not directly support loops for iterative or repetitive calculations. Brown (1999) presented a methodology to overcome this limitation, whereby a set of calculations can be repeated as many times as needed. This is achieved essentially by making the required number of copies of the equations or formulas involved. So a loop with 100 iterative calculations, for instance, would require 100 copies of the equations and the relevant parameters. Although this methodology is feasible for small, simple models, it can easily become convoluted and impractical for large, complex models, in particular those involving many equations or complex algorithms.

Another important limitation is spreadsheets like Excel do not allow one cell to alter or write to another cell. Although cells may contain formulas or functions, the effects of these formulas or functions are in situ; that is, their effects are only local or restricted to the cells that contain these formulas or functions. The contents in other cells are strictly read-only. This restriction is problematic in model building because it prevents, for instance, model parameters from being updated or initialized without having to make multiple copies of the model parameters and equations.

Consequently, the main objective of this study was to develop an Excel add-in, called BuildIt, that overcomes some of Excel's weaknesses so that Excel could be used more effectively to build crop models as well as to run model simulations. It is quite common to find agriculture models being implemented in Excel (e.g., Henriksen and Breland, 1999; Defossez et al., 2003; Theiveyanathan et al., 2004; Pathak and Wassmann, 2007). However, due to Excel's limitations, as described earlier, modelers often have to program at least some parts of their models in Excel's script language, VBA. Consequently, the challenge in this study was also to develop BuildIt that makes Excel a model building platform without requiring users to program in VBA.

It is important to stress here that the purpose of this study was neither to supplant other software modeling platforms with BuildIt, nor to compare BuildIt's features with other software platforms. BuildIt is also not a "toolbox" of mathematical and statistical routines or a collection of generic modeling components for crop growth such as components for meteorology, energy and water balance, carbon and nitrogen cycles, photosynthesis, and respiration. Instead, BuildIt attempts to improve Excel by providing it an infrastructure in which even large and practical crop models can be built and model simulations run more effectively (and without requiring users to program in VBA).

In this article we will show how BuildIt can be used to build successively more complex models in Excel. We begin with the implementation of a simple model (a leaf photosynthesis model) in Excel without the use of BuildIt, and then proceed to explain the key concepts of BuildIt by showing how the same leaf photosynthesis model can be implemented using BuildIt. Subsequently, we show the implementation of successively more complex models (canopy photosynthesis and plant growth models) to explain the more advanced features of BuildIt.

METHODS AND MATERIALS

Leaf Photosynthesis Model

Building the Model without BuildIt

Consider the following leaf photosynthesis model by Goudriaan and van Laar (1994):

$$A_L = \frac{A_m \varepsilon k I_0 \exp(-kL)}{A_m + \varepsilon k I_0 \exp(-kL)} \quad [1]$$

where A_L is the leaf photosynthesis (μg CO_2 m^{-2} leaf s^{-1}); A_m is the maximum leaf photosynthesis rate (μg CO_2 m^{-2} leaf s^{-1}); ε is the solar radiation conversion factor (μg CO_2 J^{-1}); k is the canopy extinction coefficient for solar radiation (unitless); L is the cumulative leaf area index (LAI) from the canopy top to the canopy depth being considered (m^2 leaf m^{-2} ground); and I_o is the solar irradiance above the canopies (J m^{-2} ground s^{-1}), given by (France and Thornley, 1984) as

$$I_o = \frac{I_{t,d}}{1800(t_{ss} - t_{sr})} \sin^2\left[\frac{\pi(t_h - t_{sr})}{(t_{ss} - t_{sr})}\right] \quad \text{for } t_{sr} \le t_h \le t_{ss} \quad [2]$$

where $I_{t,d}$ is the daily total solar irradiance (J m^{-2} ground day^{-1}); t_h is the local solar hour; and t_{sr} and t_{ss} are the sunrise and sunset hours, respectively.

Table 1. Three-letter codes for BuildIt actions. BuildIt actions are mathematical or cell manipulation operations not directly supported by Excel. Every action begins with a three-letter code, followed by its list of parameters; that is, *ACTION {param1:s|r} {param2:s|r} … {op:s=TRUE}*, where *ACTION* denotes a unique three-letter code of the action requested (such as ITG, UPD, and INI), and *param1, param2, …, param$_n$* are the action's parameters.

Name of action	Code	Brief description
Cumulative	ACC	Determines the cumulative sum or cumulative product of a given variable.
Arrange	ARR	Arranges the order in an array of values according to their given sequence. For example, given an array of (10, 20, 30, 40, 50) and the corresponding sequence as (3, 1, 4, 5, 2), this action would give the new array of values as (20, 50, 10, 30, 40).
Differentiation	DIF	Numerical differentiation using the three-point central (midpoint) difference method (Mathews, 1987).
Initialize	INI	Initializes one or more variables with their given initial values.
Integration	ITG	Numerical integration using the five-point Gaussian method (Mathews, 1987).
Replace	REP	Copies the values in the source cells to the destination cells.
Reverse	REV	Reverses the order in an array of values. For example, given an array of (1, 2, 3, 4, 5), reversing it would give (5, 4, 3, 2, 1).
Rotate	ROT	Rotates the order in an array of values to the left, right, up, or down direction. For example, given an array of (1, 2, 3, 4, 5), rotating it to the right or down would give (5, 1, 2, 3, 4), but rotating it to the left or up would give (2, 3, 4, 5, 1).
Run	RUN	Runs a user-defined VBA script. BuildIt supplies three utility scripts: (1) ClearOutput, (2) EnableScreenUpdate, and (3) DisableScreenUpdate. The ClearOutput script clears all model output listings from previous simulation runs. The EnableScreenUpdate and DisableScreenUpdate scripts turn on and off the Excel's screen update feature, respectively.
Shuffle	SHU	Randomly shuffles the order in an array of values. For example, given an array of (1, 2, 3, 4, 5), a shuffle might give (3, 4, 1, 2, 5), the sequence randomly determined.
Sort	SOR	Sorts the order in an array of values either ascendingly or descendingly.
Update	UPD	Updates the values of one or more variables by their rates of change using the *n*th order of the Runge-Kutta method (for solving differential equations).

	A	B	C	D	E	F	G	H	I	J
1	**PARAMETERS**			**CALCULATIONS**						
2	**Am**	1500		**th**	**n1**	**n2**	**lo**	**n3**	**n4**	**AL**
3	**e**	12		6	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D3-$B$7)/($B$8-$B$7))^2	=E3*F3	=$B$2*$B$3*$B$4*G3*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G3*EXP(-$B$4*$B$5)	=H3/I3
4	**k**	0.5		7	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D4-$B$7)/($B$8-$B$7))^2	=E4*F4	=$B$2*$B$3*$B$4*G4*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G4*EXP(-$B$4*$B$5)	=H4/I4
5	**LAI**	2		8	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D5-$B$7)/($B$8-$B$7))^2	=E5*F5	=$B$2*$B$3*$B$4*G5*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G5*EXP(-$B$4*$B$5)	=H5/I5
6	**ltd**	10000000		9	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D6-$B$7)/($B$8-$B$7))^2	=E6*F6	=$B$2*$B$3*$B$4*G6*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G6*EXP(-$B$4*$B$5)	=H6/I6
7	**tsr**	6		10	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D7-$B$7)/($B$8-$B$7))^2	=E7*F7	=$B$2*$B$3*$B$4*G7*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G7*EXP(-$B$4*$B$5)	=H7/I7
8	**tss**	18		11	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D8-$B$7)/($B$8-$B$7))^2	=E8*F8	=$B$2*$B$3*$B$4*G8*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G8*EXP(-$B$4*$B$5)	=H8/I8
9				12	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D9-$B$7)/($B$8-$B$7))^2	=E9*F9	=$B$2*$B$3*$B$4*G9*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G9*EXP(-$B$4*$B$5)	=H9/I9
10				13	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D10-$B$7)/($B$8-$B$7))^2	=E10*F10	=$B$2*$B$3*$B$4*G10*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G10*EXP(-$B$4*$B$5)	=H10/I10
11				14	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D11-$B$7)/($B$8-$B$7))^2	=E11*F11	=$B$2*$B$3*$B$4*G11*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G11*EXP(-$B$4*$B$5)	=H11/I11
12				15	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D12-$B$7)/($B$8-$B$7))^2	=E12*F12	=$B$2*$B$3*$B$4*G12*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G12*EXP(-$B$4*$B$5)	=H12/I12
13				16	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D13-$B$7)/($B$8-$B$7))^2	=E13*F13	=$B$2*$B$3*$B$4*G13*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G13*EXP(-$B$4*$B$5)	=H13/I13
14				17	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D14-$B$7)/($B$8-$B$7))^2	=E14*F14	=$B$2*$B$3*$B$4*G14*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G14*EXP(-$B$4*$B$5)	=H14/I14
15				18	=$B$6/(1800*($B$8-$B$7))	=SIN(PI()*(D15-$B$7)/($B$8-$B$7))^2	=E15*F15	=$B$2*$B$3*$B$4*G15*EXP(-$B$4*$B$5)	=$B$2+$B$3*$B$4*G15*EXP(-$B$4*$B$5)	=H15/I15

Fig. 1. Implementation of the leaf photosynthesis model without BuildIt (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

The task is to use the parameter values from Table 1 and solve Eq. [1] and [2] for hourly t_h values from sunrise (t_{sr}) to sunset (t_{ss}). Figure 1 shows one way how this task can be accomplished in Excel (without use of BuildIt). Cells in range D3:D15 hold the successive 1-hour increment from sunrise to sunset. Equations [1] and [2] are implemented in range H3:J15 and E3:G15, respectively. Because Eq. [1] and [2] are rather long equations, they are each implemented in three parts: columns E to G for Eq. [2] and columns H to J for Eq. [1].

In this example, Eq. [1] and [2] are used repeatedly to determine I_o and A_L for various values of t_h. This requires a

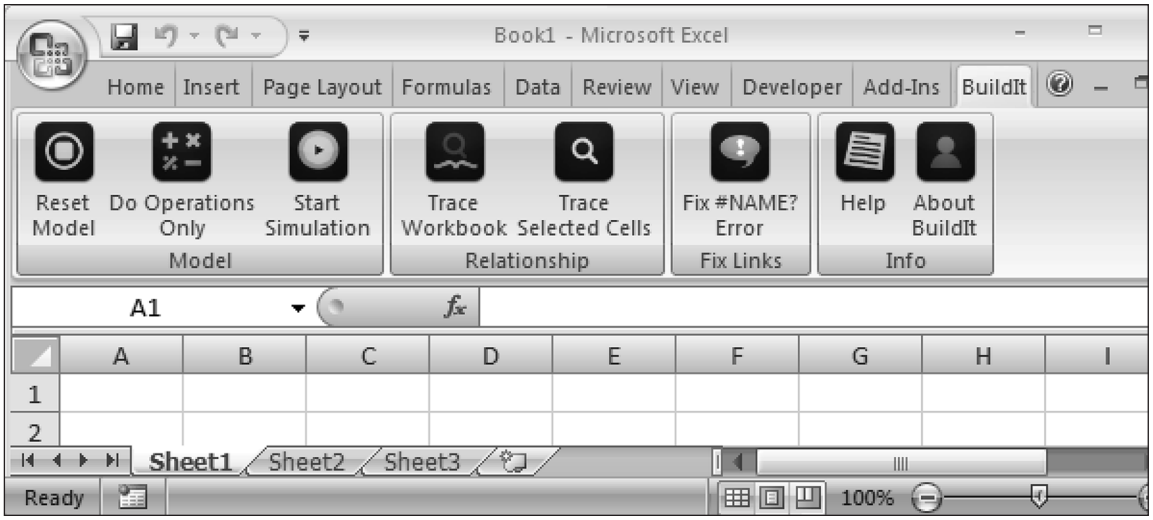


Fig. 2. **BuildIt's menu on Excel 2007 Ribbon.**

loop, which was implemented by making multiple copies of Eq. [1] and [2]. Both of these equations were first implemented in range E3:J3, then successively copied downward until range E15:J15, as only 13 iterations are required. Although this example shows how easily and quickly a simple model can be implemented in Excel, Excel's native capabilities are often limited to building simple models. It is not hard to imagine that with increasingly complex and larger models, this method of making multiple copies of equations and variables would become increasingly tortuous, impractical, and unmanageable.

Building the Model with BuildIt

BuildIt is an Excel add-in, written in Excel's script language, VBA. BuildIt works in Excel (for Windows only) version 2003 onward, and once installed, BuildIt adds its menu on Excel's main menu bar, now called Ribbon starting from Excel 2007 (Fig. 2). The pre-requisite for building models in Excel is users must be fairly competent in Excel, but experience in VBA is unnecessary. Users should at least know how to create formulas in Excel and be familiar with the list of available functions provided by Excel as well as know how to use these functions and in what situations they can be used.

BuildIt was designed to overcome some limitations in Excel by: (1) providing a loop for repetitive calculations without having users to make multiple copies of equations and variables, and (2) providing operations or tasks not directly supported by Excel (Tables 2 and 3).

Table 2. **BuildIt functions. BuildIt currently offers two functions, one for linear interpolation between two data points, and another for solving linear and non-linear simultaneous equations.**

Name of function	Brief description
Interpolate	Given an array of (x, y) values, the interpolate function performs a linear interpolation, if needed, to return the corresponding y to the given x value.
Solve	An array function to solve linear and non-linear simultaneous equations. This function uses the Newton iteration method (Billo, 2007).

BuildIt require users to set up one or more sections. A section is an area anywhere in the spreadsheet that contains information required by BuildIt or tasks to be executed by BuildIt during simulation runs. There are six sections: (1) Control, (2) Output, (3) Operation, (4) Pre-run, (5) Option, and (6) Scenario. The first two sections will be discussed first.

The Control section contains information about the loop. A loop always comprises three main components: the loop counter, the increment or interval size, and the criteria for loop continuation or termination. To obtain information about the loop, BuildIt requires users to define three cell names somewhere in the spreadsheet (Table 4): _step for the loop counter, _stepsize for the increment size, and _criteria for the criteria for terminating the loop which occurs when _criteria turns FALSE. In other words, users choose a cell to be the loop counter and define that cell with the name _step. Likewise, two other cells are chosen to be the interval size and

Table 3. **BuildIt cell names (which always begin with an underscore). Cell names _criteria, _step, and _stepsize are names given to spreadsheet cells so that BuildIt would be able to locate these cells to read their values for managing the iterative calculations. Cell names _operation, _option, and _prerun indicate the starting location of the list of BuildIt actions to be executed during simulation runs. Similarly, cell names _read and _write indicate to BuildIt what to include in the output list and where to place the output listing, respectively, during simulation runs. Lastly, cell name _scenario specifies where in the spreadsheet BuildIt would find the one or more scenarios for simulation runs.**

Cell name	Brief description
_criteria	Condition when FALSE would end the loop
_operation	Marks the start of the Operation section
_option	Marks the start of the Option section
_prerun	Marks the start of the Pre-run section
_read	Location to read in the list of model variables to be included in the output list
_scenario	Marks the start of the Scenario section
_step	Iteration step (loop counter)
_stepsize	Interval or increment size
_write	Marks the start of the output listing

Table 4. This paper shows how successively more complex model examples (leaf, canopy, then crop model) can be implemented in Excel using BuildIt. This table lists all the parameters (and their fixed values) used by these model examples.

Symbol	Description	Units	Value
A_m	Maximum leaf photosynthesis rate	μg CO_2 m^{-2} leaf s^{-1}	1500
k	Canopy extinction coefficient	–	0.50
ε	Solar radiation conversion factor	μg CO_2 J^{-1}	12
LAI	Total leaf area index	m^2 leaf m^{-2} ground	2
t_{sr}	Time of sunrise	hour	6
t_{ss}	Time of sunset	hour	18
$I_{t,d}$	Daily total solar irradiance	J m^{-2} ground day^{-1}	107
k_g	Growth rate	day^{-1}	1.10
k_d	Degradation rate	day^{-1}	0.10
k_s	Senescence rate	day^{-1}	0.05
Y_g	Efficiency of structural synthesis	day^{-1}	0.75
W_s	Storage dry weight	g C m^{-2} ground	30 (initial)
W_g	Structural dry weight	g C m^{-2} ground	130 (initial)

loop criteria, and they are given (defined) names _stepsize and _criteria, respectively. These names must be defined because BuildIt will access these cells. At the start of a simulation run, BuildIt will always initialize _step with 0 and at the end of each iteration, BuildIt increments _step by _stepsize (or $\_step_{i+1} = \_step_i + \_stepsize$) until _criteria becomes FALSE. Users should implement their models in such a way to trigger the automatic update feature in Excel to cause recalculations each time _step is changed by BuildIt.

Figure 3 and Table 5 show one way Eq. [1] and [2] can be implemented in Excel with BuildIt. As before, cells B2:B8 hold the model parameters. Cell B17, B18, and B19 are the loop's counter (_step), increment or interval size (_stepsize), and criteria (_criteria), respectively. Note that cell B17 (_step) is left blank because BuildIt will access it during simulation runs, initialising it with 0 then incrementing it by _stepsize when required. If there is any prior value (or formula) in cell B17, it will be over-written by BuildIt. Cells E7:E9 and E3:E5 implement Eq. [1] and [2], respectively. BuildIt will provide a loop for iterative calculations, so there is no need, as done previously, to make multiple copies of Eq. [1] and [2]. Cell E2 holds the hour (t_h), and it contains the formula "=_step+t_{sr}" so that simulation will begin for $t_h = t_{sr}$ (sunrise hour). Recall that BuildIt always sets _step to 0 in the beginning, and because _stepsize is 1, successive simulation run would be for $t_h = (1+t_{sr})$, $(2+t_{sr})$, $(3+t_{sr})$, and so on. Simulation stops when _criteria becomes FALSE, which, in this example would occur when t_h is greater than the sunset hour (that is, when the logical condition in cell B19 returns FALSE, which would occur when _step = 13 to give $t_h = 13+t_{sr} = 19$).

When users select "Start Simulation" from BuildIt's menu (Fig. 2), BuildIt sets _step (cell B17) to 0. This triggers an automatic update, so cell E2 (t_h) is first updated, which, in turn, updates cell E4 then E5. Because cells E7 and E8 depend on cell E5 (I_o), both these cells would next be updated. Lastly, cell E9 (AL) is updated as it uses cells E7 and E8. The update process then ends because all the dependent cells have been updated to their latest values, after which BuildIt increments _step by _stepsize. BuildIt then checks for the logical condition of _criteria. If _criteria remains TRUE, the iteration resumes, and the same cycle as described earlier repeats until _criteria becomes FALSE. At this point of discussion, there is no model output other than the values displayed in range E2:E8. To produce an output listing, users must set up the Output section.

To set up the Output section, users need to define two cell names: _read and _write (Table 4). Cell name _read indicates to BuildIt which model parameters are to be included

	A	B	C	D	E	F	G
1	**PARAMETERS**			**CALCULATIONS**			
2	**Am**	1500		**th**	=_step+tsr		
3	**e**	12		**n1**	=ltd/(1800*(tss-tsr))		
4	**k**	0.5		**n2**	=SIN(PI()*(th-tsr)/(tss-tsr))^2		
5	**LAI**	2		**Io**	=E3*E4		
6	**ltd**	10000000		**L**	=LAI		
7	**tsr**	6		**n3**	=Am*e*k*Io*EXP(-k*L)		
8	**tss**	18		**n4**	=Am+e*k*Io*EXP(-k*L)		
9				**AL**	=E7/E8		
10							
11							
12							
13							
14							
15							
16	**CONTROL**			**WHAT TO OUTPUT**			
17	**step**			**th**	**Io**	**AL**	
18	**step size**	1		=th	=Io	=AL	
19	**criteria**	=th<=tss					
20							
21				**OUTPUT LISTING**			
22							

Fig. 3. Implementation of the leaf photosynthesis model with BuildIt (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

Table 5. Cell names for the model examples. Cell names beginning with an underscore indicate BuildIt cell names. These BuildIt cell names are mandatory so that BuildIt would be able to locate the information required during simulation runs. Other cell names (without underscores) are optional but recommended because using cell names instead of their cell addresses improves formula readability and reduces risk of errors (such as mistypes).

Cell name	Cell address	Cell name	Cell address
Am	B2	AL	E9
e	B3	AT	E10
k	B4	ATd	E11
LAI	B5	Ws	E13
Itd	B6	Wg	E14
tsr	B7	_step	B17
tss	B8	_stepsize	B18
kg	B9	_criteria	B19
kd	B10	_read	D18
ks	B11	_write	D22
Yg	B12	_operation	H2
th	E2	_prerun	H7
Io	E5	_option	H11
L	E6	_scenario	H14

in the output listing, and _write indicates where BuildIt should place the output listing. In this example, _read was defined for cell D18 (Table 5). To determine what parameters would be included in the output listing, BuildIt starts to read from cell D18 and to the right (same row) onward until BuildIt encounters the first blank (empty) cell, which, in this example, is cell G18. Consequently, model parameters t_h, I_o, and A_L would be included in the output listing. Note that range D17:F17 are merely headings for the output listing. The output listing begins where the cell name _write has been defined, where in this example, _write was defined for cell D22 (Table 5), so this cell marks the start of the output listing. At the end of the first iteration (t_h=t_{sr}), the values of t_h, I_o, and A_L are written by BuildIt in the range D22:F22; the end of the second iteration (t_h=1+t_{sr}) in range D23:F23; and so on (Fig. 4). In other words, the values of t_h, I_o, and A_L are appended to the output list to finally produce the output in the range D22:F34.

This example illustrates the loop and model output facilities provided by BuildIt. Additionally, BuildIt provides actions for operations not directly supported by Excel (Table 2). Some of these actions are for specific operations such as for numerical integration, cumulative sum and cumulative product determination, initialization of variables, and the application of the Runge-Kutta method to update variables by their rates of change. Other actions are more generic such as copying the values from source cells to destination cells and manipulating cell ranges (such as rotating, shuffling, sorting, and reversing the values in the cell ranges). Without BuildIt actions, copying a value from one cell to another, for instance, would require users to directly manipulate the source and destination cells (such as using the copy-and-paste commands). However, direct manipulation of cells is not feasible during model simulation runs because, once started, model runs should continue to the end without user intervention. Additionally, no Excel functions exist that allows

	A	B	C	D	E	F	G
1	**PARAMETERS**			**CALCULATIONS**			
2	**Am**	1500		**th**	19		
3	**e**	12		**n1**	462.963		
4	**k**	0.5		**n2**	0.066987		
5	**LAI**	2		**Io**	31.01264		
6	**Itd**	10000000		**L**	2		
7	**tsr**	6		**n3**	102680.2		
8	**tss**	18		**n4**	1568.453		
9				**AL**	65.46589		
10							
11							
12							
13							
14							
15							
16	**CONTROL**			**WHAT TO OUTPUT**			
17	**step**	13		**th**	**Io**	**AL**	
18	**step size**	1		19	31.01	65.47	
19	**criteria**	FALSE					
20							
21				**OUTPUT LISTING**			
22				6	0.00	0.00	
23				7	31.01	65.47	
24				8	115.74	218.29	
25				9	231.48	381.12	
26				10	347.22	507.24	
27				11	431.95	582.92	
28				12	462.96	607.81	
29				13	431.95	582.92	
30				14	347.22	507.24	
31				15	231.48	381.12	
32				16	115.74	218.29	
33				17	31.01	65.47	
34				18	0.00	0.00	
35							

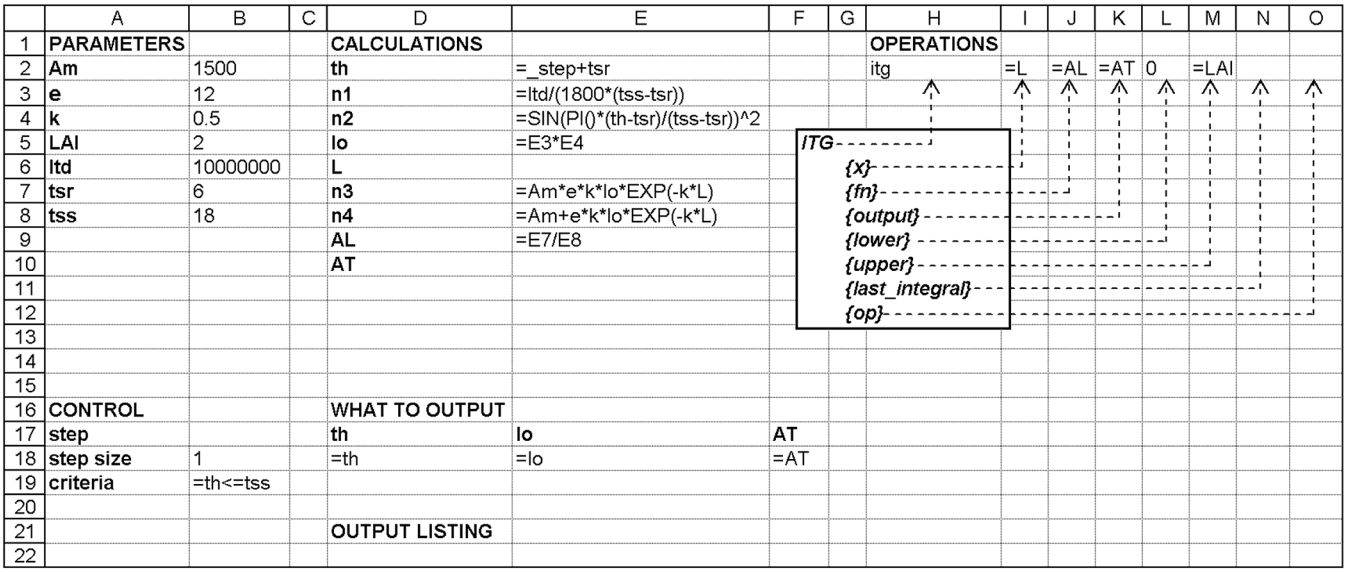
Fig. 4. Model output listing (cell range D22:F34) for the leaf photosynthesis model (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

users to change the value of non-local cells. The alternative would be to use VBA to change the values in any cells. However, this option conflicts with one purpose of this study, which was to create a model building framework in Excel that did not require users to program in VBA.

BuildIt provides 12 actions, and they must be specified strictly following their syntax. Each action has the following general syntax:

$$ACTION\ \{param_1{:}s|r\}\ \{param_2{:}s|r\}\ \ldots\ \{op{:}s{=}TRUE\}$$

where *ACTION* denotes a unique three-letter code of the action requested (such as ITG, UPD, and INI; Table 2), and $param_1$, $param_2$, …, $param_n$ are the action's parameters. The "*:s|r*" notation means a parameter can either be of type "*:s*" or "*:r*", where the symbol "|" denotes "or". The "*:s*" type means the parameter must be supplied either as a constant (such as numbers, logical values TRUE and FALSE, and text) or a single cell address (such as A1 and $B$5). The "*:r*" type is the same as "*:s*", but the "*:r*" type parameter can additionally be supplied as a contiguous cell range (such as A1:B3 and C4:C6). Note that the last parameter, *op*, for all actions is unique because this parameter is always of type "*:s*", and it accepts only logical values TRUE or FALSE. The last parameter indicates whether the given action should be executed.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**PARAMETERS**			**CALCULATIONS**				**OPERATIONS**							
2	**Am**	1500		**th**	=_step+tsr			itg	=L	=AL	=AT	0	=LAI		
3	**e**	12		**n1**	=ltd/(1800*(tss-tsr))										
4	**k**	0.5		**n2**	=SIN(PI()*(th-tsr)/(tss-tsr))^2										
5	**LAI**	2		**lo**	=E3*E4										
6	**ltd**	10000000		**L**											
7	**tsr**	6		**n3**	=Am*e*k*lo*EXP(-k*L)										
8	**tss**	18		**n4**	=Am+e*k*lo*EXP(-k*L)										
9				**AL**	=E7/E8										
10				**AT**											
11															
12															
13															
14															
15															
16	**CONTROL**			**WHAT TO OUTPUT**											
17	**step**			**th**	**lo**	**AT**									
18	**step size**	1		=th	=lo	=AT									
19	**criteria**	=th<=tss													
20															
21				**OUTPUT LISTING**											
22															

Fig. 5. Implementation of the single integration for canopy photosynthesis model (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

If *op* is TRUE, the action is executed, but a FALSE value means the action will be cancelled. This last parameter gives the flexibility of executing a given action only in certain conditions. If parameter *op* is unspecified, it is TRUE by default.

BuildIt has to be told where the actions are in the spreadsheet. For this purpose, all actions must be listed in one or more of the BuildIt sections (Table 4). However, only three sections accept actions, which are the Operation, Pre-run, and Option sections. The Operation section is most commonly used because this section is referred by BuildIt at every iteration for the list of actions to be executed. The start of the Operation section is marked by the cell name _operation. For the Pre-run and Option sections, their starting location is marked by cell name _prerun and _option, respectively. Another example will now be considered to illustrate the use of the BuildIt actions and sections.

Canopy Photosynthesis Model

Integration of Eq. [1] over the whole canopy gives the canopy photosynthesis, or

$$A_T = \int_0^{\mathrm{LAI}} A_L\,dL = \int_0^{\mathrm{LAI}} \frac{A_m \varepsilon k I_o \exp(-kL)}{A_m + \varepsilon k I_o \exp(-kL)}\,dL \qquad [3]$$

where A_T is the canopy photosynthesis (μg CO_2 m^{-2} leaf s^{-1}). In this example, the task is to use values from Table 1 to determine A_T for hourly t_h values from sunrise to sunset. Equation [3] has to be solved by integration, and BuildIt provides an action called ITG for numerical integration (Table 2).

The ITG action has the following syntax:

ITG {x:s} {fn:s} {output:s} {lower:s} {upper:s} {last_integral:s=TRUE} {op:s=TRUE}

where this action integrates the function *fn* of *x* over the interval [*lower*, *upper*], and the result of the integration will be stored in *output* cell. The second to last parameter, *last_integral*, is either TRUE or FALSE to specify if the integration operation should continue to determine the next integral. An equation with a single integral, for instance, would have one ITG action with its *last_integral* parameter set to TRUE. An equation with double integrals, however, would involve two sequential ITG actions, where the *last_integral* for the first and second ITG actions would be set to FALSE and TRUE, respectively. This is so that the first ITG action determines the first integral, followed by the second ITG action for the second integral. Accordingly, triple integrations would involve three sequential ITG actions, where the *last_integral* for the first, second, and third ITG actions would be set to FALSE, FALSE, and TRUE, respectively. Note that if omitted, *last_integral* is assumed TRUE by default. Matching ITG's parameters with the parameters from Eq. [3] means ITG's parameter *x* corresponds to the model parameter *L* in Eq. [3]; *fn* to A_L; *output* to A_T; *lower* to the value 0; and *upper* to LAI. The ITG parameter, *last_integral*, should be TRUE because Eq. [3] involves only a single integral, and the last ITG parameter, *op*, should also be TRUE because the integration should always be executed. Both these last parameters can be omitted as they would be assumed TRUE by default.

Figure 5 shows one way to implement Eq. [2] and [3]. The Operation section is marked by the cell name _operation, defined for cell H2 (Table 5). The range H2:O2 holds the specification for the ITG action. Note that the specification for any BuildIt action, including the ITG action, must be specified in contiguous cells in the same row (contiguous cell range H2 to O2, in this example). Note that the last two cells in the ITG action specification, cell N2 and O2, are blank (omitted); thus, they are assumed TRUE. ITG's parameter *output* (in cell K2) holds the value A_T, which is the cell name for cell E10. Consequently, cell E10 (AT) is the location of Eq. [3]'s numerical integration result. Cell F18 is also modified so that model parameter A_T (rather than A_L) is included in the output list.

When users select "Start Simulation" from BuildIt's menu

	D	E	F	G	H	I	J	K	L	M	N	O
1	**CALCULATIONS**				**OPERATIONS**							
2	**th**				itg	=L	=AL	=AT	0	=LAI	FALSE	
3	**n1**	=Itd/(1800*(tss-tsr))			itg	=th	=AL	=AT	=tsr	=tss	TRUE	
4	**n2**	=SIN(PI()*(th-tsr)/(tss-tsr))^2										
5	**Io**	=E3*E4										
6	**L**											
7	**n3**	=Am*e*k*Io*EXP(-k*L)										
8	**n4**	=Am+e*k*Io*EXP(-k*L)										
9	**AL**	=E7/E8										
10	**AT**											
11	**ATd**	=(12*LAI*3600*AT)/(44*10^6)										

Fig. 6. Implementation of the double integration for daily canopy photosynthesis model (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

(Fig. 2), BuildIt sets _step (cell B17) to 0. This triggers the automatic update feature to re-calculate all values in the dependent cells: first in cell E2, then E4:E5, and finally E7:E9. The spreadsheet update process then ends. At this point, cell E5 (I_o) contains the value for the solar irradiance, I_o, for the hour $t_h = t_{sr}$, but the value in cell E9 (A_L) is incorrect because cell E6 (L) does not contain the correct value (cell E6 was originally blank, so it would be taken as 0 by Excel). The value in cell E6 (L) is actually set by the ITG action to solve Eq. [3]. When the spreadsheet update process ends, BuildIt checks whether the Operation section exists. Cell name _operation was defined for cell H2 in this example, so BuildIt checks the list of actions in the Operation section (starting from cell H2) to be executed. In this case, there is only one action to be executed: the ITG action. This action uses the numerical integration using the 5-point Gaussian integration method (Mathews, 1987) and sets cell E6 (L) to five *L* values. Each setting of the *L* value triggers an automatic update, which updates values in cells E7:E9 because they depend on cell E6 (L). For each *L* value, the re-calculated value from cell E9 (A_L) is collected by the ITG action, as part of the Gaussian integration algorithm. Finally, the integration result is placed in the given location, cell E10 (A_T). Before the first iteration ends, the model parameters to be outputted are read from cell D18 (where _read was defined) to F18, after which the desired parameters (t_h, I_o, and A_T) are written in cells D22 (where _write was defined) to F22. BuildIt then increments _step by _stepsize and checks if _criteria remains TRUE. If its value is TRUE, the second iteration starts and the whole loop process as described earlier repeats until _criteria turns FALSE. After the simulation run, the whole output listing would be produced in the range E22:F34 (results not shown).

It is possible to execute the actions in the Operation section without having to go through the iterations. Consider the task to determine A_T for only $t_h = 12$. One way to accomplish this task is to modify cell B17 (_step) to have the value 6, so that cell E2 (t_h) is 12 (recall from Fig. 5 that the formula in cell E2 is "=_step+t_{sr}", and t_{sr} is set to 6 in cell B7). Then select "Do Operations Only" command from BuildIt's menu (Fig. 2). This would cause BuildIt to read the actions listed only in the Operation section without starting the loop. BuildIt executes the ITG action and places the integration result in cell E10 (A_T), giving the canopy photosynthesis of 1585.28 μg CO_2 m^{-2} leaf s^{-1} for $t_h = 12$ (result not shown). Note that executing actions via the "Do Operations Only" command would not produce an output listing.

Now consider a double integration task. Integrating Eq. [1] over the whole canopy and day gives the daily canopy photosynthesis, or

$$A_{T,d} = \frac{12 \times \mathrm{LAI} \times 3600}{44 \times 10^6} \int_{t_{sr}}^{t_{ss}} \int_{0}^{\mathrm{LAI}} \frac{A_m \varepsilon k I_o \exp(-kL)}{A_m + \varepsilon k I_o \exp(-kL)} dL \, dt_h \qquad [4]$$

where $A_{T,d}$ is the daily canopy photosynthesis (converted to g C m^{-2} ground day^{-1}). Figure 6 shows one way Eq. [4] can be solved. By selecting the "Do Operations Only" command from BuildIt's menu, the actions listed in the Operation section are read and executed sequentially (without starting the loop). The Operation section has two ITG actions to implement the double integration. The *last_integral* parameter for the first and second ITG actions are set to FALSE (cell N2) and TRUE (cell N3), respectively. The first ITG action performs the first integration, and because its *last_integral* is FALSE (cell N2), the operation continues with the second ITG action for the next integration, after which the integration process ends (because the *last_integral* for the second ITG action is TRUE). The result of the double integration is then stored in the given location, cell E10 (A_T). Cell E11 ($A_{T,d}$) completes Eq. [4] to give the result in units g C m^{-2} ground day^{-1}. In this example, the computed daily canopy photosynthesis is 22.33 g C m^{-2} ground day^{-1} (result not shown).

Program Flow

All models developed using BuildIt (including the leaf and canopy photosynthesis models discussed previously) would follow the program flow as illustrated in Fig. 7. BuildIt mediates the program flow by interjecting at certain points of the flow to guide its course. When users select "Start Simulation" from BuildIt's menu, the program flow begins by BuildIt checking whether the Option section exists by the presence of the cell name _option (Table 4). If this section exists, BuildIt reads in the list of actions in the Option section and executes them sequentially (that is, the actions are executed in the sequence they are listed). The Option section is mainly used for running user-defined or BuildIt's VBA scripts. One example of BuildIt's script is the ClearOutput script, which deletes previous output listings from past simulation runs

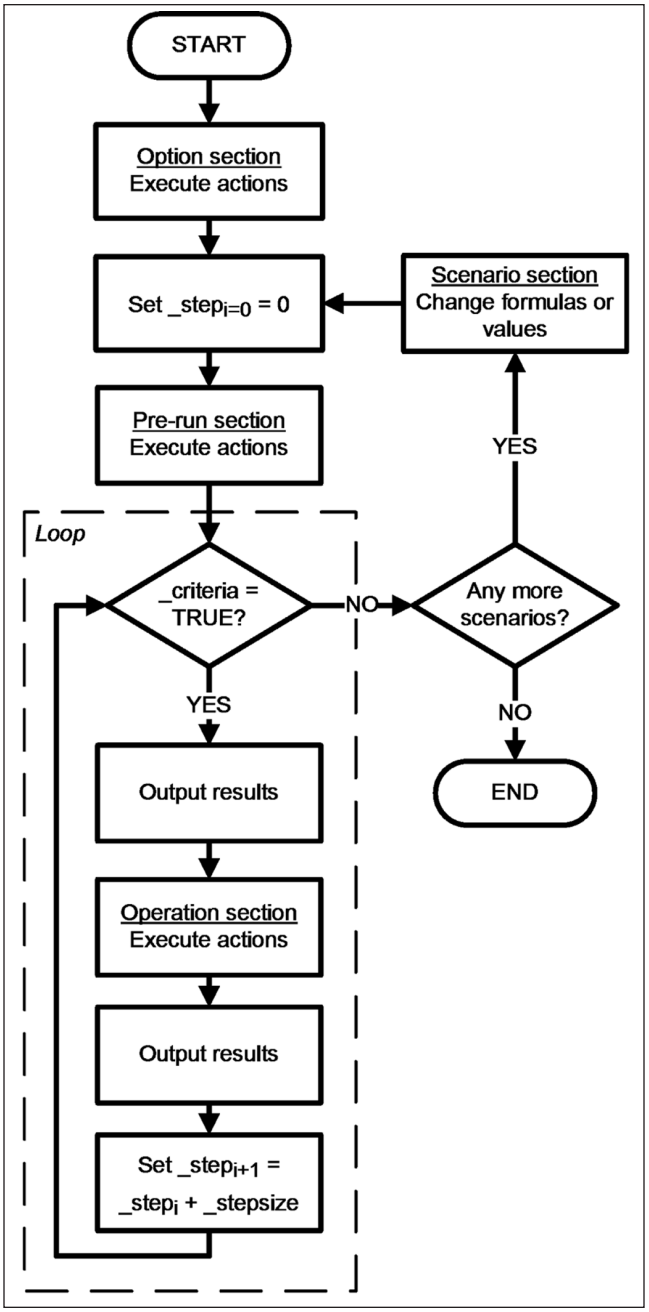


Fig. 7. Program flow of model built using BuildIt.

(Table 2 for the RUN action). Another BuildIt script is the DisableScreenUpdate script. This script disables Excel's computer screen update feature, which would often increase the speed of model runs, albeit leaving the screen to appear "frozen" or stalled during simulation runs. In short, the Option section is often used to perform "housekeeping" or preparatory tasks before the actual model calculations begin.

After the Option section, BuildIt resets _step to 0, which would trigger a spreadsheet update. After the update, BuildIt searches for the cell name _prerun to determine whether the Pre-run section exists. If it exists, BuildIt sequentially executes the actions in this section. The Pre-run section is used to execute actions just before the program flow enters the loop. Consequently, this section would be useful, for instance, to initialize model parameters (using the INI action, Table 2) such as setting the initial weight or height of a plant. Executing every action in the Pre-run section triggers a spreadsheet update. After the execution of the last action (and when the corresponding spreadsheet update ends), BuildIt resumes control and directs the program flow into the loop.

The hub of the program flow is the loop because it involves iterative calculations. The cycle of calculations and actions (listed in the Operation section) would repeat while _criteria remains TRUE. The program flow in the loop essentially involves checking the value of _criteria, executing the actions in the Operation section, outputting the results, and incrementing _step by _stepsize. This cycle repeats until _criteria becomes FALSE. Like before, executing every action in the Operation section triggers a spreadsheet update, and when the update ends, BuildIt resumes control of the program flow for the next action to be executed. After the Operation section, BuildIt increments _step by _stepsize, which would again trigger a spreadsheet update. Triggering these series of spreadsheet updates is crucial because it ensures all formulas or calculations within the loop would always use the current values of model parameters and variables.

Note that BuildIt has two opportunities to output the results: once before the program flow enters the Operation section and another when it exits the Operation section (Fig. 7). In certain cases, some model parameters might have to be outputted before the flow enters the Operation section because their values would be modified by the actions in the Operation section. In other words, their values before and after the Operation section would be different. In cases where model parameters would not be modified by the actions in the Operation section, then it does not matter if these parameters are outputted before or after the Operation section.

When _criteria turns FALSE, the program flow exits the loop, and BuildIt checks for model scenarios listed in the Scenario section. The starting cell location for this section is marked by the cell name _scenario (Table 4). The Scenario section contains new values to give selected variables or parameters. It could also contain new formulas to replace current formulas. Once these replacements are completed, the program flow resumes back to resetting _step to 0, and the program continues as described earlier. To further elaborate on this program flow, another concrete example will be used.

RESULTS AND DISCUSSION

A Simple Plant Growth Model

Consider a very simplified plant growth model adapted from Thornley (1976) (Fig. 8). The assimilates, $A_{T,d}$, produced by photosynthesis is partitioned to maintenance for the plant's survival and to growth for the synthesis of plant materials. The plant weight comprises two components: the storage weight, W_s, and the plant structural weight, W_g. The storage weight is supported by the addition of newly produced substrates via the photosynthesis process. A part of the plant's storage, $k_g W_s$, will be utilized for maintenance

and growth, where, of this total ($k_g W_s$), Y_g will be utilized for growth and the remainder (1 – Y_g) for maintenance. Of the total plant structural weight (W_g), k_d will degrade and the substrates returned to the storage, and k_s of it will senesce. The rates of change in the dry weights for storage and structure components at time t (in unit day) are thus

$$\frac{dW_s}{dt} = k_d W_g - k_g W_s + A_{T,d} \quad [5]$$

$$\frac{dW_g}{dt} = k_g W_s Y_g - k_d W_g - k_s W_g \quad [6]$$

where $A_{T,d}$ is obtained from Eq. [4]. Integrating Eq. [5] and [6] over 1 day give the amount of increase in 1 day for W_s and W_g, respectively, as

$$\Delta W_{s,t} = k_d W_g - k_g W_s + A_{T,d} \quad [7]$$

$$\Delta W_{g,t} = k_g W_s Y_g - k_d W_g - k_s W_g \quad [8]$$

Finally, the dry weights for storage and structure are updated by

$$W_{s,t+1} = W_{s,t} + \Delta W_{s,t} \quad [9]$$

$$W_{g,t+1} = W_{g,t} + \Delta W_{g,t} \quad [10]$$

where subscripts t and t+1 denote the current day and next day, respectively.

In this example, the task is to determine the daily increase in the storage and structure dry weights, W_s and W_g, over 10 days. The values for the model parameters are taken again from Table 1. Additionally two more scenarios are of interest. They are to determine, like before, the daily increase in W_s and W_g over 10 days but with: (1) a lower senescence rate, k_s, of 0.01 (instead of 0.05), and (2) a lower senescence rate, k_s, of 0.01 and a non-constant LAI, which would increase linearly with time according to: LAI = 0.1t + 2, where t is the day.

Figure 9 and Table 5 show one way these three scenarios can be accomplished. Cell range B2:B14 hold the values for the model parameters as listed in Table 1. Cell B18 (_stepsize) is set to 1 for 1-day intervals, and cell B19 (_criteria) has the formula to end the iterations after 10 days of simulation. Cell range E2:E9, like before, implements Eq. [1] and [2]. Cells E13 and F13 are defined with the names Ws and Wg (Table 5) to denote the model parameters W_s and W_g, respectively. Cells F13 and F14 implement Eq. [5] and [6], respectively.

When users select "Start Simulation" from BuildIt's menu, BuildIt checks if the _option cell name has been defined (Table 4 and Fig. 7). This cell name was defined for cell H11 (Table 5), so BuildIt reads the list of actions to be executed in the Option section (starting from cell H11). In this case, this section has only one action: the RUN action, which has the following syntax:

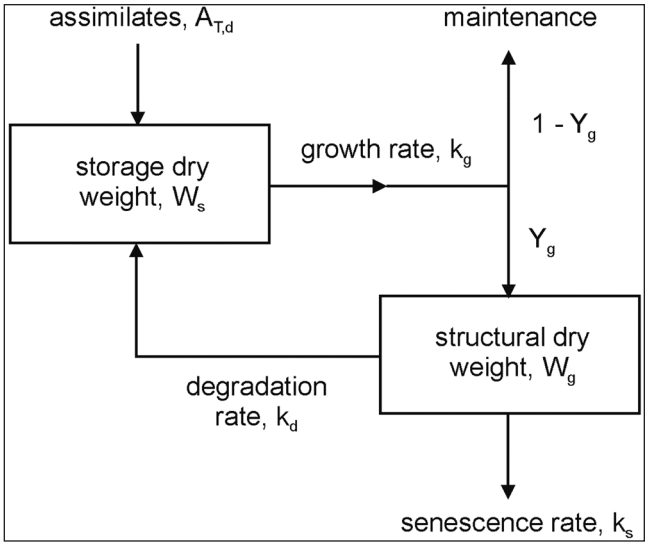


Fig. 8. Simple plant growth model adapted from Thornley (1976).

RUN {script_name:s} {op:s=TRUE}

where parameter *script_name* is the name of a VBA script (macro). In this example, BuildIt's script called ClearOutput was selected. This script clears previous output listings before actual model calculations begin. Note that the actions in the Option section are executed only once regardless of the number of modeling scenarios supplied (Fig. 7).

After the Option section, BuildIt checks for the presence of cell name _prerun, which marks the start of the Pre-run section (Table 4). The _prerun name was defined for cell H7 (Table 5), and the Pre-run section contains two actions, both of them of type INI. The INI action has the following syntax:

INI {destination:r} {source:r} {op:s=TRUE}

where this action copies the values from *source* to *destination*. This action is often used to initialize model parameters. The first INI action (range H7:K7, where last cell K7 is blank, so the *op* parameter is TRUE by default) copies the value from cell B13 to E13 (Ws). Likewise, the second INI action (range H8:K8) copies the value from cell B14 to E14 (Wg). In short, W_s and W_g are initialized to 30 and 130, respectively, by these two INI actions at the start of the simulation. Alternatively, instead of having two INI actions, they can be replaced by a single INI action that uses cell ranges: *INI {E13:E14} {B13:B14} {TRUE}*, which instructs BuildIt to copy the values from cell B13 and B14 to cells E13 and E14, respectively, in a single operation.

After the Pre-run section and provided _criteria is TRUE, the program flow enters the loop (Fig. 7). BuildIt will now determine the model parameters or variables to output before actions in the Operation section are executed. In this example, range D18:G18 in the Output section instructs BuildIt to include _step, Ws, Wg, and LAI in the output listing (Fig. 9). Notably, in one row below, both cells E19 and F19 have the value FALSE. When set to FALSE, this instructs BuildIt to output the respective variable or parameter before the Operation section (Fig. 7). If they are set to TRUE or left

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**PARAMETERS**			**CALCULATIONS**				**OPERATIONS**							
2	**Am**	1500		**th**				itg	=L	=AL	=AT	0	=LAI	FALSE	
3	**e**	12		**n1**	=ltd/(1800*(tss-tsr))			itg	=th	=AL	=AT	=tsr	=tss	TRUE	
4	**k**	0.5		**n2**	=SIN(PI()*(th-tsr)/ (tss-tsr))^2			upd	=E13:E14	=F13:F14	1				
5	**LAI**	2		**lo**	=E3*E4										
6	**ltd**	10000000		**L**				**PRE-RUN**							
7	**tsr**	6		**n3**	=Am*e*k* lo*EXP(-k*L)			ini	=Ws	=B13					
8	**tss**	18		**n4**	=Am+e*k* lo*EXP(-k*L)			ini	=Wg	=B14					
9	**kg**	1.1		**AL**	=E7/E8										
10	**kd**	0.1		**AT**				**OPTION**							
11	**ks**	0.05		**ATd**	=(12*LAI*3600*AT)/ (44*10^6)			run	ClearOutput						
12	**Yg**	0.75			**wgt**	**rate**									
13	**initial Ws**	30		**Ws**		=kd*Wg-kg*Ws+ ATd		**SCENARIOS**							
14	**initial Wg**	130		**Wg**		=kg*Ws*Yg-kd*Wg-ks*Wg		=ks	0.01						
15								=ks	0.01	=LAI	=0.1*_step+2				
16	**CONTROL**			**WHAT TO OUTPUT**											
17	**step**			**step**	**Ws**	**Wg**	**LAI**								
18	**step size**	1		=_step	=Ws	=Wg	=LAI								
19	**criteria**	=_step<=10			FALSE	FALSE									
20															
21				**OUTPUT LISTING**											
22															

Fig. 9. Implementation of the simple plant growth model (Note: Refer to Table 4 for the list of parameters. Parameters n1 to n4 are intermediaries in calculation steps).

blank (as is the case for cells D19 and G19), the output only occurs after the Operation section. Thus, the FALSE value in cells E19 and F19 instruct BuildIt to respectively output Ws and Wg before the program flow enters the Operation section. In contrast, the blanks in cells D19 and G19 (taken as TRUE by default) instructs BuildIt to respectively output _step and LAI after the Operation section.

So, before entering the Operation section, BuildIt outputs Ws and Wg (but not _step and LAI), after which BuildIt sequentially executes the actions in the Operation section (starting from cell H2 in this example; Table 5). The Operation section contains three actions, where the first two are the ITG actions for solving the double integration in Eq. [4], as described previously, to determine the daily canopy photosynthesis, $A_{T,d}$. The numerical integration result is stored in the given location, cell E10, where it is converted to units g C m^{-2} ground day^{-1} in cell E11 (ATd). This converted value is used by cell F13 to determine the rate of change in W_s, according to Eq. [5]. Cell F14, which implements Eq. [6], determines the rate of change in W_g. Both these rates of change are then used to determine the new values for W_s and W_g according to Eq. [9] and (10). For this purpose, the UPD action is used, which has the following syntax:

UPD {x:r} {rate_of_change:r} {n:s=1} {op:s=TRUE}

This action updates x by $x_{t+1} = x_t + (rate\_of\_change_t \times \_stepsize)$ using the specified nth order of the Runge-Kutta method. Note that setting the order to 1 ($n = 1$) specifies the Euler's method, which is the default method used by the UPD action.

The UPD action, specified in range H4:L4 (Fig. 9), updates cells E13 (Ws) and E14 (Wg) by their rates of change in cells F13 and F14, respectively, using Euler's method (parameter *n* in cell K4 was set to 1). After execution, cells E13 and E14 hold the new weights for the *following day*. This is why both W_s and W_g must be outputted before (not after) the Operation section. If cells E19 and F19 were set to TRUE (or left blank), the instruction would be for both W_s and W_g to be outputted after the Operation section, where their values outputted would erroneously be for the following day and not for the current day. After the Operation section, BuildIt outputs both _step and LAI. But because they are not altered by the actions in the Operation section, it actually does not matter if they are outputted before or after the Operation section; their values would remain the same either way. After their output, BuildIt increments _step by _stepsize, and the previously described cycle repeats until _criteria turns FALSE. The output listing is displayed in range D22:G32 (Fig. 10).

When _criteria becomes FALSE, BuildIt exits the loop and determines whether there are more model scenarios to run (Fig. 7). BuildIt checks for the existence of the cell name _scenario for the Scenario section (Table 4). The Scenario section, however, does not accept any actions. Instead, a scenario must be specified as an array of (*destination*, *new_values_or_formulas*) paired entries. These paired entries must also be in contiguous cells and in the same row. In Fig. 9, range H14:I14 instructs that, in the second scenario, cell B11 (ks) will have a new value of 0.01 (cell I14). The program flow then continues as shown in Fig. 7. The output listing for the second scenario is displayed in range D35:G45 (Fig. 10). In the third scenario, there are two pairs of (*destination*, *new_values_or_formulas*) entries (range H15:K15 in Fig. 10). The first pair instructs, like before, that cell B11

	D	E	F	G
16	**WHAT TO OUTPUT**			
17	**step**	**Ws**	**Wg**	**LAI**
18	11	65.32	297.92	2
19		FALSE	FALSE	
20				
21	**OUTPUT LISTING**			
22	0	30.00	130.00	2
23	1	32.33	135.25	2
24	2	32.62	141.63	2
25	3	33.23	147.30	2
26	4	33.73	152.62	2
27	5	34.21	157.55	2
28	6	34.66	162.15	2
29	7	35.08	166.42	2
30	8	35.46	170.39	2
31	9	35.82	174.09	2
32	10	36.15	177.53	2
33				
34	Scenario no. 2:			
35	0	30.00	130.00	2
36	1	32.33	140.45	2
37	2	33.14	151.67	2
38	3	34.18	162.33	2
39	4	35.14	172.67	2
40	5	36.08	182.67	2
41	6	36.99	192.34	2
42	7	37.86	201.69	2
43	8	38.71	210.74	2
44	9	39.53	219.50	2
45	10	40.32	227.96	2
46				
47	Scenario no. 3:			
48	0	30.00	130.00	2
49	1	32.33	140.45	2.1
50	2	35.09	151.67	2.2
51	3	37.94	163.94	2.3
52	4	40.93	177.20	2.4
53	5	44.05	191.47	2.5
54	6	47.29	206.75	2.6
55	7	50.66	223.02	2.7
56	8	54.15	240.29	2.8
57	9	57.76	258.53	2.9
58	10	61.49	277.75	3

Fig. 10. **Output listing for all the three scenarios for the simple plant growth model.**

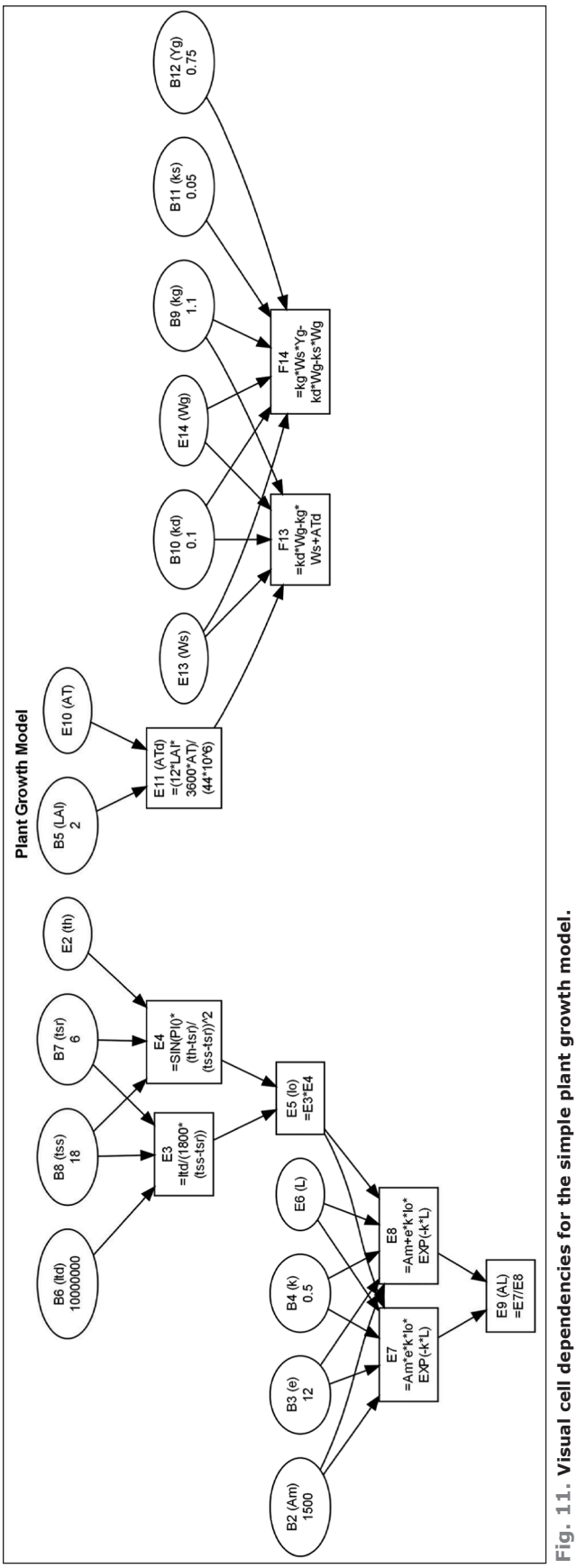


Fig. 11. **Visual cell dependencies for the simple plant growth model.**

(ks) will have a new value of 0.01 (cell I14). The second pair instructs that cell B5 (LAI) will no longer be a constant like before but be determined by the formula "=0.1*_step+2." After completing the third scenario, the output listing is displayed in range D48:G58 (Fig. 10). When all scenarios have been exhausted, BuildIt ends the simulation run.

Figures 9 and 10 show that no more than 100 spreadsheet cells were required to implement the plant growth model for the third scenario (the count included and excluded the cells used for output listing and labeling, respectively). BuildIt reduces spreadsheet complexity and produces spreadsheet models that are more concise and maintainable. Without BuildIt, the implementation of the same model for the third scenario, for instance, required nearly 2700 cells (implementation not shown). This count is a 27-fold increase in the number of cells, most of which were for implementing the Gaussian integration method and for the iterative calculations.

BuildIt provides a visual tracing tool. It visually depicts the cell inter-relationships by drawing a diagram (called a directed graph) as an aid to understand the overall model computations as well as to help in debugging during the model building process. BuildIt translates the various cell relationships into a DOT-language (Koutsofios and North, 1992) script, which is then interpreted by Graphviz (Gansner and North, 2000) to render the directed graph. This visual tracing tool is invoked by choosing "Trace Workbook" or "Trace Selected Cells" from BuildIt's menu (Fig. 2). The difference between the two commands is "Trace Workbook" visually depicts all cell relationships in one or more worksheets in the current workbook, whereas "Trace Selected Cells" only depicts the relationships of selected cells in the current worksheet.

The implementation of the plant growth model (Fig. 8 and 9) is visually depicted as a diagram by BuildIt in Fig. 11. Ellipse nodes denote cells containing values or text, whereas rectangle nodes denote cells containing formulas. For example, cells B6, B7, B8, and E2 are depicted as ellipse nodes because they contain values (or blanks such as for cell E2). Their corresponding cell names are shown in brackets. Cell E3 (which has no name) is a rectangle node because this cell contains a formula, and it depends on or requires information from cells B6, B7, and B8. Consequently, an arrow is drawn from each of these nodes to node E3.

A Complex Plant Growth Model

BuildIt can be used for larger, more complicated models such as the daily generic crop (DGC) model described by Thornley and France (2004) in their book. However, because the DGC model comprises approximately 70 equations with more than 170 parameters, this article will only give an overview of how BuildIt was used to implement this model in Excel. The DGC model comprises four core components or submodels: environment, plant, litter and soil, and hydrology.

The environment submodel contains information regarding the sowing date and seeding rate, site location, nitrogen deposition, and N fertilizer rates and application dates. This submodel also calculates the daily meteorological properties (such as the minimum and maximum air temperature, rain, relative humidity, wind speed, and net radiation) based on sine functions. Although these meteorological properties are calculated, they can be substituted with measured or real weather data. The measured weather data, for instance, can be arranged as a table in a separate worksheet, and the Excel table function, VLOOKUP, can be used to look up the meteorological properties for a given date or day of year. The DGC's plant submodel essentially determines the daily changes to the weights of shoot, roots, and product (such as fruit, seeds, and tubers) as well as the C and N substrates in the shoot and roots. To determine these weights, other properties must be first calculated such as the leaf area index, plant height, interception of solar radiation, plant N uptake, litter fluxes, and C and N transport between the shoot and roots. This submodel also considers the affects of air temperature and water stress on plant growth. The crop development process (phenology) is represented by three stages: germination, anthesis, and maturity, where each successive development stage is reached when a critical amount of thermal units is accumulated. The litter and soil submodel determines the amount of litter, soil organic matter, and nitrogen in the soil. Lastly, the hydrology submodel calculates the amount of soil water after taking into account the evapotranspiration and net rainfall. Overall, the DGC model consists of 11 state variables: seven in the plant submodel (weights of shoot, roots, and product, and the weights of C and N substrates in the shoot and root), three in the litter and soil submodel (weights of soil litter, soil C, and soil N), and one in the hydrology submodel (amount of soil water).

The DGC model was implemented using 12 worksheets in a single workbook. For instance, the worksheet named "Environment" contained the formulas to determine the daily meteorological properties, and the "Photosynthesis" worksheet for the leaf area index, plant height, solar radiation interception, and net photosynthesis. The "Phenology" worksheet kept track of the current plant development stage by the use of three flags to represent the germination, anthesis, and maturity stages. These three flags would be toggled from 0 to 1 when the plant successively reached germination, anthesis, then maturity. These flags were toggled using Excel's logical function, IF-THEN-ELSE, to determine if the current accumulated thermal units exceeded the critical thermal units for a given developmental stage. The "Growth" worksheet contained the formulas to determine the growth rates for the shoot and roots as well as the C and N transport between the shoot, roots, and product. The "Weights" worksheet contained the initial weights for the shoot, roots, product, and the C and N substrates in the shoot and roots. Their initial weights were calculated based solely on the seeding rate. The "CN" worksheet contained the formulas to determine the litter, C, and N fluxes. Two worksheets, named "Hydrology" and "ET," were used to calculate the evapotranspiration, net rainfall, and amount of soil water, as well as the water stress functions such as the reduction due to water stress on stomatal conductance, photosynthesis, and C and N fluxes. The "Table" worksheet does not have any formulas. Instead two tables were placed here: one table for rainfall interception according to month (empirically determined) and another table for listing the values for

latent heat of vaporization of water (λ), saturated air vapor pressure (SVP), and psychometric constant (γ) at various air temperatures (values from the second table are used in the Penman-Monteith calculations to determine the evapotranspiration). The values from these two tables were obtained by using either Excel's VLOOKUP function or BuildIt's function, interpolate (Table 3). For instance, given the month, the VLOOKUP function returns the corresponding value of the rainfall interception. Likewise, given the air temperature, interpolate function returns the corresponding values for λ, SVP, and γ, using linear interpolation if required. The "Rates" worksheet contained DGC's 11 state variables and their respective calculated rates of change. The "Output" worksheet contained the output listing. Both _read and _write cell names were defined for cells in this worksheet so that model output would be located in a dedicated worksheet, separated from the model calculations and user input.

The last worksheet is "Control." It contains the loop information for BuildIt: _step, _stepsize set to 0.1 for 0.1-day increments, and _criteria set to end the simulation when plant maturity was reached, as indicated when the maturity flag toggled to 1. Day of year (DOY) was calculated from _step, using the formula given by Thornley and France (2004), so that _step = 0 gives DOY =1, _step = 364 gives DOY = 365, but _step = 365 returns DOY to 1. A Pre-run section was set up to contain several INI actions to initialize the 11 state variables and three phenology flags (all flags initialized to zero). The Operation section was set up to contain the UPD action, which would update the 11 state variables with their respective rates of change using the fourth order Runge-Kutta method. Although not required, the Option section was set up to contain one RUN action to execute the ClearOutput script to remove any previous output listing.

When users choose "Start Simulation" from BuildIt's menu, the program flow begins by reading the Option section for any actions to be executed before the simulation run (Fig. 7). The _step is then reset to zero. This triggers a spreadsheet update so that all the dependent values are re-calculated. The Pre-run section is next read, and all the 11 state variables and three phenology flags are initialized. Again, this triggers a series of updates to re-calculate all the dependent values. The _criteria is checked, and if its value remains TRUE, the loop begins. Model parameters (such as the 11 state variables and leaf area index) are outputted before BuildIt reads the Operation section. In the Operation section, the UPD action is executed to update the 11 state variables with their respective rates of change. After the UPD action, their updated values are for the following day (which would be outputted in the next iteration before the Operation section). After the Operation section, the remaining model parameters are outputted. BuildIt then increments _step by _stepsize, and the whole loop process described above repeats until _criteria turns FALSE.

Using BuildIt, the model implementation required no more than 320 spreadsheet cells (not including the cells used for output listing and labeling). It is conceivable that, without BuildIt, the implementation of this large model (which involves 70 equations and 170 parameters and many repetitive calculations) would not only require far more spreadsheet cells but also produce a spreadsheet model that is more complex and more difficult to maintain. The Excel workbook implementing this DGC model using BuildIt can be obtained from the corresponding author.

EXCEL LIMITATIONS

BuildIt overcomes some limitations of Excel, but other notable limitations persist. First, Excel models often run slower compared with models developed in other software platforms. This is partly because models developed in other platforms, such as in C++ or Fortran, are compiled into fast machine code that is in a format directly executable by the computer. In contrast, no such compilation occurs in Excel. Slower execution speeds may not be noticeable for simple spreadsheet models, but for larger, more complex models (due to their numerous and more intricate calculations), their execution speeds can be noticeably sluggish and become a significant constraint.

Second, it is difficult, if not impossible, to support model reusability and extendibility in Excel without resorting to VBA. These two concepts are related to ways to increase the usefulness, applicability, and lifespan of models. A modeling framework that adheres to these two concepts means that, ideally, the framework is flexible and adaptable enough for algorithm modification, substitution, and addition. Reusability and extendibility are analogous to the "Plug-and-Play" feature in modern computer systems. For instance, adding, replacing, or removing a computer peripheral (such as a printer or scanner) should not cause hardware malfunction or software conflicts. Similarly, a reusable and extendible model means that one or more parts in the model can be modified or even substituted with parts from another model. It also means that new features or functions can be added to the model. All these operations can occur without having to break or redesign the existing modeling framework. Examples of agriculture frameworks developed with concerns of achieving reusability and extendibility are such as by Acock and Reddy (1997), Caldwell and Fernandez (1998), Hillyer et al. (2003), Papajorgji et al. (2004), Papajorgji and Pardalos (2006), and Moore et al. (2007).

Third, it is easier to introduce errors into spreadsheet models compared with models developed elsewhere. Excel's lack of modularity, structure, and validation mean models can be built easily and quickly. However, these properties also encourage bad programming practices, especially by novices or non-programmers. These bad practices, in turn, encourage models that are difficult to understand and maintain as well as create models that contain errors. Scoville (1994) remarked, "Spreadsheets make it easy to do complex calculations—and even easier to do them incorrectly." Nevertheless, the spreadsheet modeling guidelines from Read and Batson (1999), Raffensperger (2003), and Kruck (2006) are particularly useful to eliminate the risks of errors as well as to produce spreadsheet models that are easier to understand and maintain.

Fourth, Excel does not provides tools to directly help in model development such as performing Monte Carlo simulations, helping in model parameterization and calibration,

evaluating model performances, and profiling models (such as identifying code bottlenecks). However, one of Excel's strength is its versatility, which means that these tools can be implemented in Excel either by methods advocated from literature [such as Barreto and Howland (2006) and Guerrero (2010) for Monte Carlo simulations] or the use of commercial Excel add-ins.

CONCLUSION

BuildIt provides an infrastructure in Excel for building crop models and running simulations more effectively and without requiring users to program in VBA. By overcoming some weaknesses in Excel, BuildIt allows modelers to benefit from Excel's ease of use, flexibility, and power in model building and running simulations. BuildIt reduce spreadsheet complexity and produce spreadsheet models that are more concise and maintainable. BuildIt is free, and it can be downloaded from http://www.christopherteh.com/buildit (verified 16 May 2011).

ACKNOWLEDGMENT

This study was funded by the Research University Grant Scheme, Universiti Putra Malaysia (Research no. 01-01-09-0692RU). Powersim is a registered trademark of Powersim Corporation; Stella is a registered trademark of isee systems, inc.; ExtendSim is a registered trademark of Imagine That Inc.; Maple is a registered trademark of Waterloo Maple Inc.; Mathematica is a registered trademark of Wolfram Research, Inc.; Microsoft Excel and Visual Basic are registered trademarks of Microsoft Corporation; and OpenOffice.org is a registered trademark of Team OpenOffice.org e.V.

REFERENCES

Acock, R., and V.R. Reddy. 1997. Designing an object-oriented structure for crop models. Ecol. Modell. 94:33–44. doi:10.1016/S0304-3800(96)01926-6

Barreto, H., and F.M. Howland. 2006. Introductory econometrics. Using Monte Carlo simulation with Microsoft Excel. Cambridge University Press, Cambridge.

Billo, J.E. 2007. Excel for scientists and engineers: Numerical methods. John Wiley & Sons, New Jersey.

Brown, A.M. 1999. A methodology for simulating biological systems using Microsoft Excel. Comput. Methods Programs Biomed. 58:181–190. doi:10.1016/S0169-2607(98)00077-7

Caldwell, R.M., and A.A.J. Fernandez. 1998. A generic model of hierarchy for systems analysis and simulation. Agric. Syst. 57:197–225. doi:10.1016/S0308-521X(96)00071-6

Defossez, P., G. Richard, H. Boizard, and M.F. O'Sullivan. 2003. Modeling change in soil compaction due to agricultural traffic as function of soil water content. Geoderma 116:89–105. doi:10.1016/S0016-7061(03)00096-X

France, J., and J.H.M. Thornley. 1984. Mathematical models in agriculture. Butterworths, London.

Gansner, E.R., and S.C. North. 2000. An open graph visualization system and its applications to software engineering. Softw. Pract. Exper. 30:1203–1233. doi:10.1002/1097-024X(200009)30:113.0.CO;2-N

Goudriaan, J., and H.H. van Laar. 1994. Modeling potential crop growth processes. A textbook with exercise. Current issues in production ecology. Kluwer Academic, the Netherlands.

Guerrero, H. 2010. Excel data analysis. Modeling and simulation. Springer, Berlin.

Henriksen, T.A., and T.A. Breland. 1999. Evaluation of criteria for describing crop residue degradability in a model of carbon and nitrogen turnover in soil. Soil Biol. Biochem. 31:1135–1149. doi:10.1016/S0038-0717(99)00031-0

Hillyer, C., J. Bolte, F. van Evert, and A. Lamaker. 2003. The MODCOM modular simulation system. Eur. J. Agron. 18:333–343. doi:10.1016/S1161-0301(02)00111-9

Khandan, N.N. 2002. Modeling tools for environmental engineers and scientists. CRC Press, Boca Raton, FL.

Koutsofios, E., and S. North. 1992. Drawing graphs with Dot. Technical report. AT&T Bell Laboratories, Murray Hill, NJ.

Kruck, S.E. 2006. Testing spreadsheet accuracy theory. Inf. Softw. Technol. 48:204–213. doi:10.1016/j.infsof.2005.04.005

Mathews, J.H. 1987. Numerical methods for computer science, engineering, and mathematics. Prentice-Hall, Englewood Cliffs, NJ.

Moore, A.D., D.P. Holzworth, N.I. Herrmann, N.I. Huth, and M.J. Robertson. 2007. The Common Modelling Protocol: A hierarchical framework for simulation of agricultural and environmental systems. Agric. Syst. 95:37–48. doi:10.1016/j.agsy.2007.03.006

Nardi, B., and J.R. Miller. 1990. The spreadsheet interface: a basis for end user programming. p. 977–983. *In* D. Diaper et al. (ed.) INTERACT 90: 3rd IFIP International Conference on Human-Computer Interaction. North-Holland, Amsterdam.

Papajorgji, P., W.B. Beck, and J.L. Braga. 2004. An architecture for developing service oriented and component-based environmental models. Ecol. Modell. 179:61–76. doi:10.1016/j.ecolmodel.2004.05.013

Papajorgji, P.J., and P.M. Pardalos. 2006. Software engineering techniques applied to agricultural systems: An object-oriented and UML approach. Springer Science+Business Media Inc., New York.

Pathak, H., and R. Wassmann. 2007. Introducing greenhouse gas mitigation as a development objective in rice-based agriculture: I. Generation of technical coefficients. Agric. Syst. 94:807–825. doi:10.1016/j.agsy.2006.11.015

Raffensperger, J.F. 2003. New guidelines for spreadsheets. Int. J. Business Econ. 2:141–154.

Read, N., and J. Batson. 1999. Spreadsheet modelling good practice. IBM & Institute of Chartered Accountants in England and Wales, London.

Scoville, R. 1994. Spreadsheets. The PC Bible. Peachpit Press, Berkeley, CA.

Seila, A.F. 2005. Spreadsheet simulation. p. 11–18. *In* M.E. Kuhl et al. (ed.) Proceedings of the 2005 Winter Simulation Conference. IEEE, California.

Theiveyanathan, S., R.G. Benyon, N.E. Marcar, B.J. Myers, P.J. Polglase, and R.A. Falkiner. 2004. An irrigation-scheduling model for application of saline water to tree plantations. For. Ecol. Manage. 193:97–112. doi:10.1016/j.foreco.2004.01.025

Thornley, J.H.M. 1976. Mathematical models in plant physiology. Academic Press, London.

Thornley, J.H.M., and J. France. 2004. Simple generic daily crop model. p. 388–413. *In* Mathematical models in agriculture. Quantitative methods for the plant, animal and ecological sciences. 2nd ed. CABI, UK.