MSR128

Developing a Checklist to Increase Efficiency of VBA Code

Medland S[1], Davies H[1], Butler K[1]

[1] York Health Economics Consortium, Enterprise House, Innovation Way, University of York, York, YO10 5NQ

BACKGROUND AND OBJECTIVES

Several health technology assessment bodies specify a maximum economic model run time in their pharmacoeconomic guidelines [1, 2]. However, as models become larger and more complex, the run time increases. Hence, it is increasingly important that models are run efficiently. For models built in Microsoft Excel, Visual Basic for Applications (VBA) code is frequently used to run sensitivity analyses and calculate complex model data. Efficient writing of VBA code is associated with efficient running of the model.

This research aims to give advice on the writing of bespoke and efficient VBA code.

METHODS

A model with a slow deterministic sensitivity analysis (DSA) was reviewed by a modeller proficient in VBA code. Areas where the code efficiency could be improved were identified and used to develop a corresponding checklist (Table 1). The checklist was subsequently used to rewrite the code, whilst maintaining an identical output. The length of code and the average run time of the original and new code were compared.

Table 1: VBA code checklist

Check to be conducted	Why is this important?
Limiting interaction between VBA and Microsoft Excel	
Does the code refer to a particular model cell or range? Repeated navigation to the value can increase interaction and slow the model.	The cell value or range should be assigned to a defined variable. The VBA can make changes to / use the defined variable and, later, this can be output in the model as appropriate.
Are the following codes used? ▪ ".select" ▪ ".copy" ▪ ".pastespecial"	Use of these codes should be avoided as it can cause repeated navigation in the spreadsheet. These can be replaced by named ranges and defined variables, which can complete the same actions in a shorter time.
Are there references to: "activesheet" or "activecell"?	These can cause errors if the macro is run from a different worksheet. The worksheet or range should be referred to specifically. For example, Sheets("SheetName") and Range("RangeName").
Has "Application.WorksheetFunction" been used?	Only VBA functions should be used where possible.
Are calculations switched to manual while the code is running?	This prevents unnecessary, time-consuming calculations in the background. To set calculations to manual, use: Application.Calculation = xlCalculationManual
Are worksheet updates switched off while the code is running?	This prevents the worksheet updating unnecessarily when the VBA code makes changes. To stop the worksheet updating as the code runs, use: Application.ScreenUpdating = False
Are the results outputted line-by-line?	A defined variable can be used as a results array in VBA code. This gathers all the results and outputs them in a specific range once the analysis is complete.
Reducing code length and time for reviews or updates	
Is the same task applied to a set of values?	Writing bespoke code for each value is timely and results in long code that is difficult to review and update. *For loops* can be utilised so that the same piece of code is run over each value in the set. This will help to shorten the code length, thereby facilitating review and shortening the time taken to adapt the model in the future.
Is a specific task repeated several times across the same or different subroutines? For example, collecting the results from the model.	Create a custom public function or subroutine to be used within a module. This helps to shorten the code length and, therefore, facilitate review.
Have ranges been hardcoded in the VBA code? Avoid hardcoding ranges into the VBA.	If a row or column is added in the VBA, the range of interest will shift, and can cause errors in the workbook. Avoid hardcoding the ranges. Instead, cell ranges used in the VBA should be named appropriately to allow for ease of updates and increase transparency.
Is option explicit defined at the top of the module? Write "option explicit" at the top of the module.	Option explicit should always be used to force the modeller to define all variables. This helps to prevent spelling errors in variable names causing issues in the code.
Formatting	
What variable types have been used?	The choice of variable type is important. *Strings* and *doubles* should be used instead of *variants* of arrays because they take up less storage space in the VBA (10 bytes and 8 bytes, respectively, compared to 16 bytes) [3].
Have Microsoft Excel-specific features been used in the front-end?	Conditional formatting and other Microsoft Excel functions, including offset functions and what-if analyses, should be avoided.
Have modules been named appropriately?	This helps to facilitate review
Have appropriate comments been added to the code?	This helps to facilitate review

RESULTS

- The average length of the new code was 89.7% shorter than the original code (Figure 1).
- The average run time for the new code was 78.2% shorter than the original code (Figure 2).
- While the original code was running, the VBA continually referred to the worksheet to identify the value of each hard-coded range and to paste each result as the scenario was run. Use of "activesheet", "activecell", and ".select" added an unnecessary interaction between the code and the workbook and slowed the model. Using defined variables also decreases interaction between the worksheet and the code and, therefore, run time. Other ways to reduce interaction include avoiding ".select", ".copy" and ".pastespecial", switching off worksheet updates and only calculating the model when necessary.
- Utilising loops, rather than writing bespoke code for each task, shortened the code length. If a task is repeated in the code, custom functions and subroutines can be used within a module and cell ranges from the workbook can be assigned to a defined variable. Option explicit should always be used to ensure all variables are defined to prevent errors.
- The choice of variable type is important: *strings* and *doubles* take up less storage space than *variants* [3].
- After development of the VBA code checklist, calculations were set to automatic and the screen updating was not switched off, so the worksheet updated as the code ran. Switching calculations to manual, only calculating the model when necessary and switching off worksheet updates contributed to a shortened run time.

Figure 1: The change in average code length after actioning the VBA code checklist

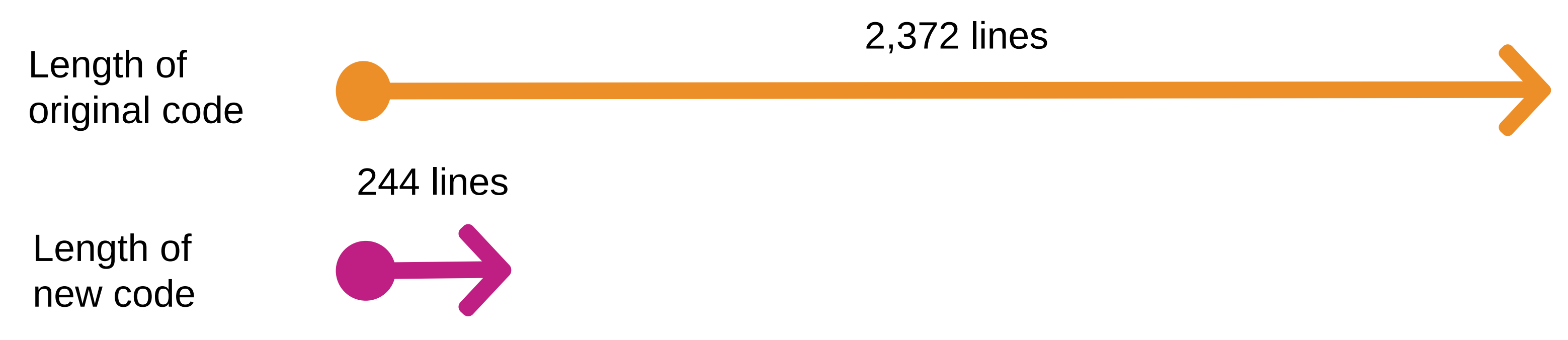
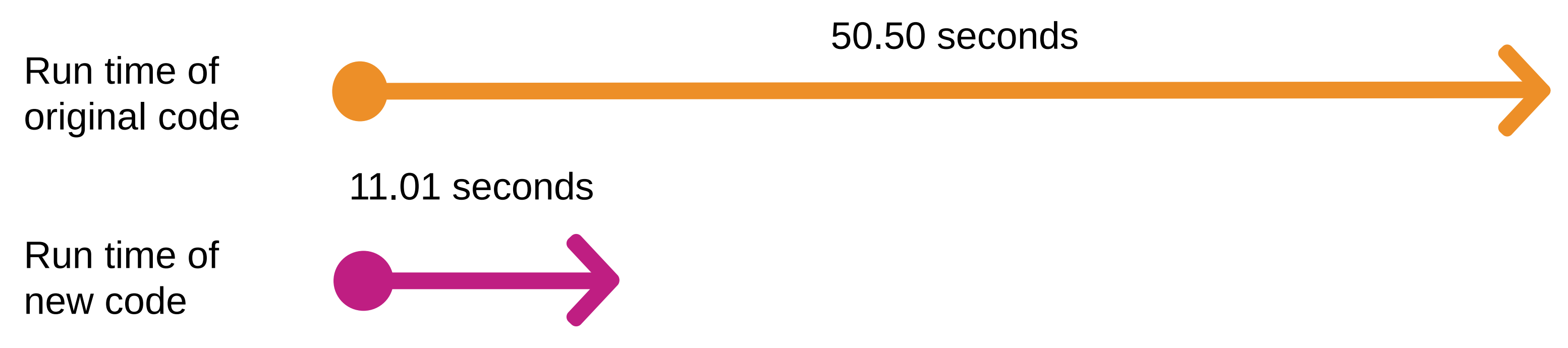


Figure 2: The change in average DSA model run time after actioning the VBA code checklist



CONCLUSION

The increasing size and complexity of economic models [4] requires efficient use of VBA code. The use of the principals outlined in this poster can reduce the time taken to conduct and adapt an analysis and facilitate review by increasing readability and reducing code length.

REFERENCES

1. CADTH Reimbursement Review: Economic Requirements Checklist. Canadian Agency for Drugs and Technologies in Health (CADTH). March 2022. **2.** PBAC guidelines: Section 3A.9 Uncertainty Analysis. Pharmaceutical Benefits Advisory Committee (PBAC). September 2016. **3.** Visual basic for applications: Data type summary. Microsoft. 2023. **4.** Poirrier, J.E., et al. Up to new limits: the number of petaflops necessary for realistic health economic models. 2023.

CONTACT US

sarah.medland@york.ac.uk

Telephone: +44 1904 322278

Website: www.yhec.co.uk

http://tinyurl.com/yhec-facebook

http://twitter.com/YHEC1

http://tinyurl.com/YHEC-LinkedIn

UNIVERSITY of York

INVESTORS IN PEOPLE® We invest in people Gold

YHEC York Health Economics Consortium