Basic Semi-Automated Testing

Anders Balari
vba-flow.net

Table of Contents

Introduction 2
Why Would We Want to Test? 3
Now It's Your Turn: Think About Benefits and Pains 4
Reviewing Your Turn: Benefits and Pains 5
Basics About Testing 7
Now It's Your Turn: Write Some Tests 9
How Does the Basic Semi-Automated Testing Work? 10
Now It's Your Turn: Rebuild the Demo, Code Your Tests 16
Test Driven Development 20
Now It's Your Turn: Try Test Driven Development 22
More Powerful Testing with Rubberduck or FF2 23
About VBA.Flow() and Flow Framework 2 27
Outro 29

Introduction

Excel VBA only does have a rudimentary possibility for tests: Debug.Assert - when using it, code execution will pause at the assertion if it fails. That's better than nothing, but much worse than what is really useful.

With this small solution you can easily write and run tests when working with Excel VBA. It is not a test suite, it is not fully automated, but it is small and easy to use, with the potential to add a lot of value to your development experience.

You can find this solution in the subfolder https://github.com/gueleh/vba-flow/tree/main/basic-semi-automated-testing.

If you already decided that you do want to test and also know the basics about testing, you may want to go directly to the explanation of the solution: How Does the Basic Semi-Automated Testing Solution Work?

Why Would We Want to Test?

Automated testing is the state of the art in modern software development. If something is the state of the art, there are some pretty good reasons for this[1].

Face it: your software already is always tested. If you don't do it, the users do it. But except for intentional testing they do it unintentionally while they use your solution to produce their work results.

If the users don't catch an error, it is passed on to the next stage, which again is likely to contain unintentional testing or even intentional testing, e.g. by the auditor who audits financial statements or tax returns.

So why not do the testing yourself, making sure that you only deliver high quality solutions that do what they're supposed to do without errors?

Automated testing adds benefits and removes pains. Which ones? Let's do something similar to so called "closed book learning"[2].

---

[1] Or someone does have a pretty good marketing department, based on which someone rules the world in terms of what is considered to be the state of the art.

[2] Closed book learning means coming up with answers related to what you want to learn before you have learned it. You: WTF? Me: Calm down, it's a cool thing. Closed book learning might sound counter-intuitive, but it is your own effort and also the errors you most likely will make that actually do make all the difference. It turned out that learning and retaining stuff is easier when you first came up with wrong results and find this out by comparing your results with the contents of your text book afterwards.

Now It's Your Turn: Think About Benefits and Pains

Having in mind what you read in the last section, please take a pencil and a sheet of paper.[3] Please also do use a timer.

Which pains result from not testing software solutions?

- Write down the ideas you have about this.
- Also write down your personal experiences.
- Set the timer to 3 minutes.
- And go for it!

Which benefits result from testing software solutions?

- Write down the ideas you have about this.
- Also write down your personal experiences (even if you didn't test your solutions this far, I am pretty sure that as a user you do have experience with well tested and badly tested software).
- Set the timer to 3 minutes.
- And go for it!

[3] I'm not kidding you. Yes, a pencil and a sheet of paper. This is another well proven thing about learning: writing or drawing with your hands significantly improves your learn success. It is generally a good thing to do: when I am designing solutions for Excel VBA, I work with paper prototypes and also do jot down a lot of my code design per hand - my experience is that this improves my creativity and clarity, improves my overall cognitive performance. There's also evidence for this, in case you're asking.

Reviewing Your Turn: Benefits and Pains

What are the key insights and ideas you had during the exercise?

We already were thinking about the fact that your Excel VBA solutions are always tested, one way or the other. In the worst case by creating some damage and finding out about this.

This could be solved by setting up a sufficient test plan for all of the business results your solution is supposed to produce. This plan then could be tested manually, either by yourself or by a dedicated user.

While this already would be better than doing intentional testing during production, it on one hand is a manual activity and on the other hand takes place after your development work. This is neither convenient nor efficient.

Correct results with a tool users love to use is one thing. And then there are new or changed requirements, there also might be regular maintenance.

Whenever you change something in the structures relevant for correct execution of your solution, it is somewhat probable that you break something.

Wait a moment, why didn't I write "whenever you change something in the code"? Because in Excel many times structures that are not VBA code are involved, e.g. cell contents, worksheet structure, defined names etc.

Even if your architecture, your code and all the other relevant structures are totally clean, it is somewhat probable that you break something.

If you do use automated testing and do have a sufficient test coverage, you will quickly know when you broke your solution by adding, changing or removing something.

Adding tests for sure is something you do in addition, i.e. it needs additional time. When assessing this, the additional effort has to be weighed against the time, nerves or even business damage you save due to having sufficient automated testing in place.

A Quick Retrospective Regarding the Exercise

How did it feel to use pencil and paper?

How did it feel to use a timer and the so called time-boxing?

Basics About Testing

What is testing? Giving a very basic definition, a test is the comparison of an expected result of an activity with the actual result in order to find out, if these two results match. If they don't match, some follow-up activities are likely to happen in order to fix the situation.

Tests are somewhat integrated in our everyday life. How's that? Think of you fetching cash from a cash dispenser. When you enter the amount you do want to withdraw, pick up the cash the dispenser gives you and check whether it is the amount you requested, you do a test. Actually, a lot of other tests are involved in the process of fetching cash from a cash dispenser, with a minimum of two actors: the cash dispenser and you.

So, usually grown-ups do have a pretty good natural and experience-based understanding about testing. It is even fair to say that it is a built in feature and that without this feature our species would have ceased to exist a long time ago.

Take a couple of minutes if you wish, thinking about all the bad things that could happen to a living being not being capable of doing tests all the time, many of them even being automated tests in a way.

But hey, let's get a bit more formal and specific. When designing and documenting tests, following information can be useful:

- a test ID for quick reference
- a description of the test, explaining what it tested plus everything else that might be required to understand what it is all about

- the starting state, i.e. which preconditions have to be met
- the input parameters used to do the test
- the expected output
- the actual output
- the test result, e.g. "passed", "not passed" or even "not executable" as a third possible result
- in case of more than one test: an overall summary, at least indicating whether there are failed tests

This represents the data fields used by the "Basic Semi-Automated Testing" solution for Excel VBA. Below you find a screenshot showing the test log for the demo tests of the solution.

	A	B	C	D	E	F	G
1	Tests	6	Passed	5	All Passed?	FALSCH	
2							
3	ID	Description	Input	Expected Output	Actual Output	Starting State	Passed?
4	1	sTransformToUpperCase	Anders Balari	ANDERS BALARI	GUENTHER LEHNER	None.	FALSCH
5	2	sTransformToUpperCase	Anders Balari	ANDERS BALARI	ANDERS BALARI	None.	WAHR
6	3	Intentional execution error of non-trivial function	Args causing division by zero	False (for execution result)	Falsch	None.	WAHR
7	4	Non-trivial function execution error does not change ByRef return value	n/a	"INF" (for ByRef return value, standing for "infinity")	INF	Function already executed with intentional error.	WAHR
8	5	Correct execution of non-trivial function	Args not causing an error	True (for execution result)	Wahr	None.	WAHR
9	6	Correct division result	lValueOne = 10; lValueTwo = 2	dAchievedResult = 5	5	Function already executed without error.	WAHR

Now it's time for an exercise that will also produce value for one of your Excel VBA solutions.

Now It's Your Turn: Write Some Tests

Same drill: please take a pencil and a sheet of paper. This time I leave it up to you whether you want to use a timer or not[4].

You are going to write test cases in this exercise, using the fields you saw in the screenshot in the last section.

I recommend that you do this for one of your Excel VBA solutions. This then would not only be a learning experience, but at the same time a first step toward automated testing of your solution.

If you don't want to do it for one of your solutions, or even in addition: think of everyday situations, like the example with the cash dispenser, write test cases for them and also experience-based results, if possible. Hey, perhaps this even uncovers bits and pieces of your everyday life that would benefit from you doing intentional tests.

If you go for writing test cases for one of your Excel VBA solutions, please take as much time as you want to invest. Otherwise, please set your timer to 15 minutes.

And remember: write down the stuff on a piece of paper, using a pencil.

---

[4] Well, you could always try the so called "pomodoro technique" if you're not familiar with it. It is highly beneficial to our well-being and thus also to our work results, for a whole bunch of reasons the explanation of which would finally reach out far beyond the scope of this document. The idea, in a nut shell: Set a timer to 25 minutes. Work 25 minutes with high focus. Set a timer to 5 minutes. Do a 5 minute break, ideally standing up, moving around etc. Then repeat. (I was no friend of doing this while developing - how could this ever work, with me being interrupted during this wonderful idea about the xyz-functionality etc. etc.? Actually, it works fine, leading to even more wonderful ideas).

How Does the Basic Semi-Automated Testing Work?

In medias res[5].

If you haven't already done so, please go to the repository[6] and download the demo workbook and the two class files from https://github.com/gueleh/vba-flow/tree/main/basic-semi-automated-testing.

The solution does consist of two classes[7]:

- i_C_Test.cls -> representing a single test
- i_C_TestLogger.cls -> used to log test results to a worksheet

In the demo workbook, you can see how these classes are used. More about that in the section below.

The Class i_C_Test.cls

This class is used to document one test per class instance. It stores the required fields:

- [Test] ID (string)
- [Test] Description (string)

---

[5] A good exercise for learning what clean code is, would also be to critically review this guidance document and identify any "unclean code" you can find in my writing. No, I am not kidding you. So, if you're in, take a note somewhere and make sure you'll be aware of it when you are going to learn about clean code - either by reading "Clean Code" by Robert C. Martin or by reading a future VBA.Flow() nugget of value that is going to arrive in your in inbox eventually.

[6] Even though Git does not directly support version control for Excel workbooks, it can be done rather easily to a certain extent. This is going to be covered in a future free VBA.Flow() nugget of value.

[7] If you are not familiar with using class modules in Excel VBA yet: fear not, I got you covered. In this guidance document and in the demo workbook you are going to find everything you need to use the testing solution. Plus, this will give you a first impression on what you can do with classes.

- Input [Parameters] (string)
- Expected Output (string)
- Actual Output (string)
- Starting State (string)
- Test Passed (boolean)

Why strings? This gives you a lot of flexibility on how you use these fields, e.g. in case of more than one input parameters or in case of giving an explanation for the expected or actual output along with the value.

The fields are private. The values are set with subs and retrieved with functions as follows:

- Sub AddTest(string, string, string, string, string, optional boolean, optional string)
  - all fields can be set by passing in all parameters when calling it
  - the boolean test result and the string with the actual output are optional parameters.
- Sub SetResult(boolean)
  - sets the test result
  - you don't need this sub if you set the result already when calling AddTest
- Sub SetActualOutput(string)
  - sets the actual output achieved with the tested code
  - you don't need this sub if you set the result already when calling AddTest
- Function vaGetTestData() as Variant
  - returns a two-dimensional array (base 1) containing all fields, so that its contents easily can be written into a Range with just one statement (see below for an example)
  - one row and seven columns, with the column order like in the field list exhibited above, i.e. (1 To 1, 1 To 7) in the declaration

    - this function is used by i_C_TestLogger.cls to easily retrieve the field values
    - thus you don't need this function unless you want to read the field values yourself
    - printing the contents to a Range is very easy, e.g. Range("A1:A7").Value2 = vaGetTestData()
- Function bPassed() as Boolean
    - returns the result result if needed separately (e.g. for handling failed tests in the code automatically)

Please refer to the section "How To Use the Solution" below for a suggestion on how to use the class.

The Class i_C_TestLogger.cls

This class is used to write the test results and a basic summary to a worksheet. Please note that the worksheet contents are completely replaced each time, starting with cell A1.

The class has only one sub: LogTestResults(worksheet, collection)

- "worksheet" takes a Worksheet object[8], informing the class about the worksheet in which the test results are supposed to be logged
- "collection" takes a Collection object[9], containing the instances of the i_C_Test class, which are supposed to be logged

[8] If you do not know yet how to work with Worksheet objects, please search for this in the web. It's not hard to grasp. The easiest way in this case would be just to pass in the desired worksheet with its CodeName (being an important concept for clean code in Excel - please look it up if necessary)

[9] If you do not know yet how to work with Collection objects, please search for this in the web. It's also not hard to grasp. Collections are very powerful and a crucial part of working with Excel VBA professionally. Most likely you already did, e.g.

Why is a worksheet object one argument of this sub?

- This is called "encapsulation": the class does not need to know about which worksheet to be used, until it is told when calling LogTestResults.
- This makes it easy to use the class in other workbooks.
- It allows you to use more than worksheet for test logging in bigger projects.

Please refer to the section "How To Use the Solution" below for a suggestion on how to use the class.

The Demo Workbook BasicSemiAutomatedTesting.xlsm

In this workbook you find an example for how I normally do use this testing solution in module "pM_TestingDemo".

You should be able to understand everything based on the code and the comments.

Please note that the comments also contain additional information which I deemed to be potentially useful and relevant, even though they are not required to use the testing solution.

This is also true for the module "pM_CodeToTest", which does contain the tested code, but also some guidance related to some basic concepts of my VBA.Flow() approach - not the "real/full thing", but some key aspects that can add a lot of value to your way of working already.

In the project explorer of the VBA editor (often referred to as "VBE") you can see the three worksheets of the demo

---

ThisWorkbook. Worksheets is a collection and with ThisWorkbook.Worksheets(1) you access its first element.

workbook. If you are not used to working with worksheet code names (Worksheet.CodeName), please note the code names I defined for these worksheets and how I used them in the code, e.g. when using the class "i_C_TestLogger" to log the test results.

The worksheet "README" contains a brief documentation of the essential information on the solution, including installation instructions[10].

In the next chapter there will be an exercise, in which you rebuild the demo usage and/or create code for testing one of your own solutions. Before we go there, let's have a look at a generic guidance.

How To Use the Solution

To use the solution,

- you need a sub with declarations of the two classes[11] plus one collection.
- you need a worksheet for logging the test results.
- For each test
  - you create a new instance of i_C_Test,
  - set the test documentation
  - and add the instance to the collection
- At end you call "LogTestResults" of your i_C_TestLogger instance to log the test results

[10] Please note: My Excel 365 installation currently does have a strange glitch. When I import class modules, they are imported as normal modules and the code shows the header of .cls files, which normally is invisible. I fix this by renaming the imported class, creating an empty class module, rename it accordingly and copying the contents of the imported class to the new class (without the normally invisible header lines).

[11] You only need one instance of the logger class, so you might want to instantiate it directly in the declaration statement, i.e. Dim oC_Logger as New i_C_TestLogger, in comparison to Dim oC_Test as i_C_Test for the test class, for which you need more than one instance.

With pseudo code:

```
Dim oC_Test as i_C_Test
Dim oC_Logger as New i_C_TestLogger
Dim oCol_Tests as New Collection

'Test 1
Set oC_Test = New i_C_Test
oC_Test.AddTest string, string, string, string, string,
  boolean[12], string
oCol_Tests.Add oC_Test

'Test 2
Set oC_Test = New i_C_Test
oC_Test.AddTest string, string, string, string, string,
  boolean, string
oCol_Tests.Add oC_Test

' etc.

oC_Logger.LogTestResults wksTestLog, oCol_Tests
```

[12] If possible, I set the test result by doing the actual test. So, if the test is to find out, whether ExpOutput equals ActOutput, my parameter for the test result when calling .AddTest would be "ExpOutput = ActOutput".

Now It's Your Turn: Rebuild the Demo, Code Your Tests

You know the drill. But hey, this time it's different. Instead of improving our pencil and paper capabilities, we do want to write actual code in Excel VBA.

This exercise consists of two parts. You should at least do one of them.

- The first part: rebuild the demo code, i.e. the sub "mTestingDemo()" in module "pM_TestingDemo" of the demo workbook.
- The second part: write the code for testing one of your own solutions[13].

Part One: Rebuild the Demo Code

Remember "closed book learning"? If not, please go to chapter Why Would We Want to Test? and read the footnote referred to in the last sentence of the chapter.

Let's do it this way:

- study the sub "mTestingDemo()", (the "pre closed book phase")
- create a new module with one sub (starting the "closed book phase")
- and rebuild the tests which are done in the demo sub,
- including the logging.
- Then create a new worksheet,
- run your code
- and compare the test log with the one created by the demo code. (the "open the book again" phase)

[13] If you did this with pencil and paper in the last exercise, now's the time to find out about your current pencil and paper capabilities by transforming your effort into working code.

Please note: you don't have to do it exactly in the way it is done in the demo, as the demo shows different ways of using the solution.

If you not already did it this way, I invite you to try the pomodoro technique (please go to chapter Now It's Your Turn: Write Some Tests and read the footnote referred to in the first sentence, if this doesn't ring a bell).

Did it work? If yes, time to fetch the bubbly and put it into the fridge for later (or to fetch whatever you might prefer over the bubbly, in my case a nice non-alcoholic "Hefeweizen" beer).

Part Two: Write the Code For Testing One Of Your Solutions

Again, I invite to apply the "closed book learning" approach, like in part one of this exercise.

Let's do it this way:

- Select one of your own Excel VBA solutions
- Import the two classes
- Create a worksheet to log the test results
- If it does make sense, create some test data, e.g. by setting up and filling one or more worksheets or by doing it in any other way you see fit
- Create a module for the sub or function[14] you do want to use for testing and logging the results
- Uh huh, we arrived at a fork:
  - Either write the code for the tests you already defined in the prior exercise "Write Some Tests"

[14] Function? Yes, in case you do want to do it as a "non-trivial lower level function" with error handling, like it is exhibited in the code to be tested in the demo workbook.

    - or select the functions and subs of your solution you want to test and write the code for the tests
- Write the code to log the test results
- Run the tests, best do it after each new test you add

This is my last invitation (at least in this document) to try the pomodoro technique (refer to part one of the exercise, if this doesn't ring a bell)

Did it work? If yes, time to go to the fridge, where your bubbly is already waiting for you, or *your* nice non-alcoholic "Hefeweizen" beer (hopefully not mine, otherwise we got some' to sort out).

Time for a Review

Hey, now you know how to use my basic semi-automated testing solution! I hope that it is useful for you!

Let's answer some questions:

- What were your key insights?
- How can this help you from now on?
- What will be the specific next step if you decide to use this? Define it right now, so that you keep the momentum.
- Which feedback do you want to give me, regarding the solution? What can be improved? What can be added?
- Which other solutions or topics are you interested in, on your learning journey to Enlighten Your Way of Working with Excel VBA?

Time for a Retro

Now let us visit the meta level:

- How was your learning experience? What did you like about it, what did you dislike?
- Which of the suggested techniques did you try, which not? Why, why not?
- How do you feel about these techniques?
- How can I improve future learning experiences? What did you like about the guidance? What did you dislike?

Test Driven Development

There's a bit to say about Test Driven Development ("TDD"),

- its benefits,
- the pains it removes
- and the pains it causes (at least in Excel VBA),
- its bigger context,
- its being considered as a standard practice of really professional software development by some iconic people like Robert C. "Uncle Bob" Martin
- and a lot more.

All of this would be beyond the scope of this document. However, let us at least have a brief look.

TDD is a practice consisting of two steps, which are repeated at a high frequency.

Step 1: write a failing test for the next bit of code you are going to write.

Step 2: write the code to pass the test.

This can be a very healthy and rewarding practice, which lets you flow through your day with fully tested new code, almost like breathing. It is reported that there are developers that have a frequency of less than one minute for each step, i.e. one minute to write the failing test, then one minute to write to code to pass the test etc. etc.

What do I do? It depends.

Sometimes I feel like applying TDD in my Excel VBA work, especially when I create a lot of smaller

functionalities, which don't take a lot of time to be coded.

And then again, I don't apply it.

In most of my projects I do work alone and there's not a high frequency of integration of the work of different people into one working piece of software. In these cases I decide freely how to handle it in this specific project.

I invite you to a last exercise.

Now It's Your Turn: Try Test Driven Development

Let's keep it brief:

- open a project you currently work on
- set up all you need for testing like you learned recently
- select some new functionalities you want to add to your project
- for each of these,
  - set up a sufficient skeleton which you can test, e.g. just a function and its arguments, with no code in it
  - write a failing test
  - write the code to pass the test
  - move on to the next functionality
- do this for a while, e.g. for one hour
- assess how this feels for you

More Powerful Testing with Rubberduck or FF2

Given the name of this testing solution, it is hopefully not much of a surprise for you that I myself consider this solution to be basic and semi-automated.

While this already might add substantial value for you, you might want to use more powerful solutions.

As mentioned in the introduction, Excel VBA does not support real testing natively. But there are ways, two of which I want to mention in this final chapter.

Rubberduck[15]

Rubberduck is an add-in for the "VBE", i.e. the VBA code editor. It is free and open source, however the authors are happy about anyone funding them a coffee or buying some of the wonderfully mad Rubberduck merchandise.

The test suite is one of several useful features and I strongly recommend that you at least give all of them a test drive.

There's also the "Rubberduck Style Guide", a useful document that also provides guidance for some important concepts of software development - partly overlapping with some of the future free VBA.Flow() nuggets of value.

When I started using Rubberduck, I was enthused. This changed quickly and now I do not use it very often. This is why:

[15] First being slightly irritated by the name, I now love it and do consider it to be brilliant: I remembered it immediately, together with the solution behind it.

- some of my projects are quite big, making Rubberduck crash and with it the whole Exel instance,
  - sometimes when opening a second somewhat big project
  - sometimes even when just working with one big project
- this might have to do with my using the add-in "MZ Tools", however if I'd have to choose between only using one of them, my decision wouldn't even take one second: MZ Tools.
- the VBE is a really crappy editor in comparison to other IDEs, which also extends into the realm of possible performance of add-ins, at least based on the smart guys who developed Rubberduck:
  - the add-in observes the whole code
  - for the sake of performance, this is done not in "real time", but in a lower frequency,
  - requiring the user to press the "refresh" button from time to time
  - and significantly reducing the VBE performance during a refresh cyle (and this is the decisive pain for me)

But, nonetheless it's a fine add-in, which I still use from time to time, which is why I decided to mention it here.

You can find Rubberduck here: https://rubberduckvba.com/.

Flow Framework 2

"Flow Framework 2" is my own framework for developing solutions with Excel VBA. In the future it also will have a testing suite that will be more powerful and more automated than the independent solution described in this document.

I do use "Flow Framework 2" almost every time when I create a new solution with Excel VBA.

Almost?

- I don't use it for very small solutions, especially if they are only used for a very limited time or even once
- I don't use it for solutions I inherited from other people, unless budget and life cycle let the migration or refactoring be a viable or even necessary option
- I don't use it when it does make more sense to use its predecessor "Flow Framework", which only is an option for very big solutions that might not be suitable for Excel to begin with, plus it's too limiting to really be a framework

Let us look into the past and into the future.

The Past

"Flow Framework", the predecessor mentioned a minute ago, does contain a somewhat mature and powerful test suite. Unfortunately, it has some minor flaws and a major encapsulation problem -> it would take significant effort to migrate it to "Flow Framework 2" and to extend it with what I plan to do in...

The Future

One thing I disliked in the old test suite was that it was not possible to test stuff going on inside of the tested components.

For "Flow Framework 2", I came up with an approach which might solve this - it then will be possible to "x-ray" during the tests.

However, my philosophy for Flow Framework 2 is to develop only what really is needed, at the time it really is needed often enough. And I did not need the more powerful testing suite often enough until now.

Thus, while the current version already has some required basic structures for the future testing suite, it does not have a testing functionality yet.

Thus, too, I happily do use the "Basic Semi-Automated Testing Solution" until the testing suite in "Flow Framework 2" will be developed.

You can find Flow Framework 2 here: https://github.com/gueleh/flowframework2

Please note: the current state of the framework does not have the level of guidance required for other persons learning how to use it step by step. This is going to be added over time, along with adding the already planned features.

Enlighten Your Way of Working with Excel VBA

Learn how to boost stakeholder satisfaction with the "VBA.Flow()" approach and the optional usage of the free[16] "Flow Framework 2", allowing you to create solutions that both users and developers love to work with.

By sharply increasing efficiency, maintainability, extensibility, robustness, reusability and collaborativeness, you are going to take your overall development experience to a whole new level.

Start Your Free Learning Journey Now

Subscribing to the free VBA.Flow() newsletter will bring valuable contents to your inbox regularly.

No advertising, only free nuggets of value that will help you on your learning journey.

---

[16] Commercial usage of "Flow Framework 2" might require a paid license, subject to individual agreement.

Visit https://vba-flow.net/ and subscribe to the newsletter now, you won't regret it.

May the Cell be With You

Anders

Outro

If you want to give me feedback, you're invited to get in touch.

If you liked or even loved this part of the journey, please spread the word to other people who might be interested.

https://www.linkedin.com/in/andersbalari/

https://www.linkedin.com/showcase/vba-flow/