DATE: 4/20/20

AUTHOR: Noah B

TITLE: Use cases of the Budget Tracking Tool

OVERVIEW

The question of implementing a Budget Tracking tool is essentially a question of comparing the contents of two separate Excel files: a weekly report produced from Bluebeam Markups, and the estimate worksheet. These two files also imply an order – the estimate file must exist before the report file can be compared to it.

Upstream and downstream factors are also important. Upstream issues address the technological/behavioral factors that go into producing a weekly report that contains (at least) the critical fields.

Downstream issues are concerned with processing the results of the comparison once it has been successfully performed.

Having considered several solutions, I wish to make the case that a database-centered approach is the best option for long term sustainability. This solution should also utilize VSTO/C# (over Excel VBA) as the preferred implementation for reasons that will be outlined below.

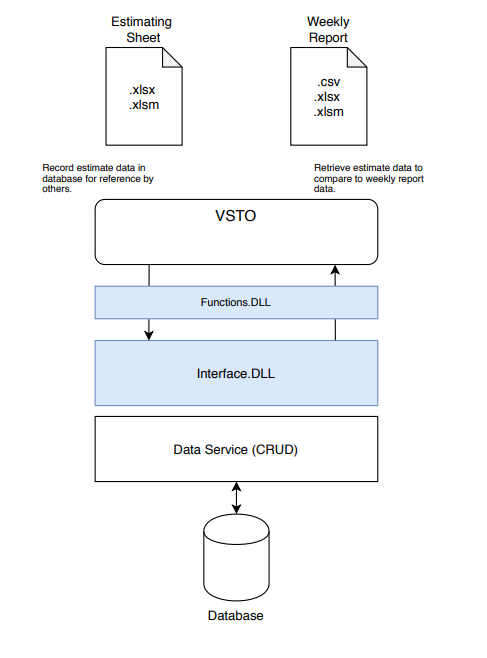
DISCUSSION

The database-centered architecture is shown in *Diagram 1*. It is comprised of the following components:

1. Database (or area in existing database) to hold output data from the Estimating sheet.
2. VSTO Add-In that provides an option for recording data from the Estimate sheet into the database, and another option for processing an incoming report from the field.
3. A Functions library that hold the code that interacts with the database.
4. An Interface library that holds the interface requirements for the functions and data objects.

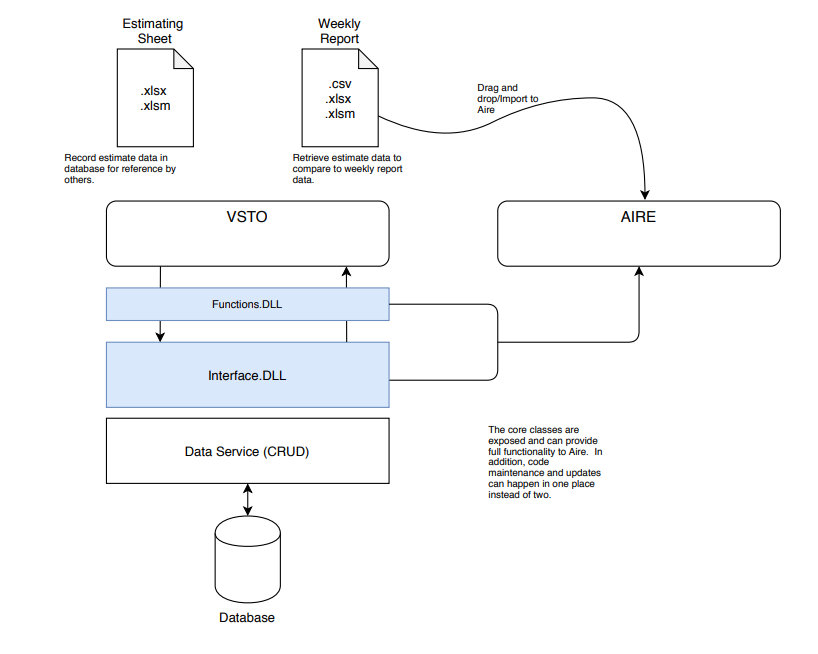
The best feature of this arrangement is that the code to interact with the data objects has been decoupled from the VSTO, whose primary concern will be the gritty details of getting information FROM (but not “TO”) the Excel sheets. Once the raw data has been gathered, the VSTO references the code libraries to create the data objects that will be processed.

Having decoupled the core functionalities, another selling point of this architecture can be seen. *Diagram 2* shows how Aire can make use of the code libraries.



***Diagram 1***

Since Aire supports drag-and-drop functionality, and since the file formats are included in Aire’s list of “importable formats”, it can easily support a drag-drop import of the weekly report. Because the estimating data has been recorded in a central location (the database), any application can reference it to make the desired comparison. This would not be the case with a “power-spreadsheet” implementation unless the VBA included database support.[[1]](#footnote-1)



***Diagram 2***

One last feature to note is that this arrangement eliminates the possibility of duplicate code. Suppose a VBA-endowed workbook is created and, in the future, it is decided that Aire will support easy importing of weekly reports. This will require duplicate code, since Aire cannot access any of the existing VBA. Further, when changes are made to the overall design (someone decides they want to put additional information in a report, etc.), the code must be maintained in two places instead of one. The decoupled database-centered architecture solves these problems.

These are just the arguments for the architecture; there are also some technological considerations. Among the most relevant are:

* VSTO has access to the extensive .NET framework, VBA does not.
* C# supports object-oriented design, but VBA has severe limitations in this regard. The main significance of this is that separate libraries cannot be created using VBA[[2]](#footnote-2) and therefore the decoupling on which the architecture depends could not be implemented.

To summarize, this decoupled architecture should be used because:

1. It is easily maintained and easily modified.
2. It is accessible to Aire and any other application that wishes to interface with it.
3. (not mentioned above) The data will be stored for the long term, where it has the potential to be meaningfully analyzed in the future if needed.

In addition, a .NET-based solution like (but not limited to) VSTO should be chosen because:

1. The tool will have access to the .NET framework and has the potential to be extended in the future.
2. VBA cannot be used to implement the decoupled architecture because of its inherent limitation in supporting object-oriented design.
3. The deployment of VSTO is cleaner and less prone to user-complications than a macro-enabled workbook or a VBA-based Excel Add-In.

USE CASE

The user actions are also allowed to be decoupled from each other[[3]](#footnote-3). As a matter of fact, it’s the presence of the database that makes this possible, since once the estimating data has been recorded any subsequent actions no longer need to locate or open the Estimating sheet ever again. If changes need to be made to the data, the Estimating dept person can simply open the sheet, update the data, and commit the changes to the database by using the VSTO feature. By committing the data to the database, the Estimating sheet is effectively no longer a dependency of the subsequent steps.

When a weekly report is opened in Excel, the user selects the appropriate action in the Add-In to process and compare the report data. The functions of importing the data and converting it to the designed data objects is performed by the VSTO by using features of the standalone libraries, which also support printing and routing of the data.

SUMMARY

I hope it’s clear that the decoupled architecture shown, along with the choice of a .NET solution like VSTO are the best choices for developing the Budget Tracker. You’re probably wondering about the size and complexity of the tool at this point. Believe it or not, the size and complexity are negligibly larger than would be a VBA-based solution, which would still have to implement the responsibilities of the program, but without the benefits conferred by the .NET alternative.

The differences are, almost without exception, differences of how to organize the source code for the program, NOT the actual size of the source code which, as I have mentioned, would be roughly the same number of lines or possibly less if the .NET option is used. VBA code will get surprisingly large in short order, especially because it does not have access to the abundant resources of the .NET Framework.

Please see the attachment “Development Plan” (next pages), which outlines the concrete responsibilities of the development team.

DATE: 4/20/20

AUTHOR: Noah B

TITLE: Budget Tracker Development Plan

ABSTRACT: Outlines responsibilities for the development team.

The development plan is as follows:

SETUP

* Set up the new project and add it to Source Control for version management.
* The functional and interface libraries should be combined into one class library project, titled “Estimating.ProgressReporter”. The directory structure for the project should include the folders “Interfaces” and “Services”; the interfaces for Services should be located in their own folder inside the “Interfaces” folder.
* The services will need a client to interact with the database; add a project and call it “Estimating.SQLService”; this will contain the implementation for the objects to interact with the database.
* Add the VSTO project and call it “Estimating.VSTO”.

BUILD

* Create first drafts of the comparator logic; discover what the actual data needs are compared to the design.
* Using the discovered needs, create/revise the interfaces/services in ‘Estimating.ProgressReporter”.
* Implement the interfaces as service classes in “Estimating.ProgressReporter”.
* Hand off to Mike D; at this point, he should have working service classes and access to the interfaces for both services and data objects. His primary focus will be in routing and formatting the spreadsheet data so that it interfaces correctly with the services.

DEPLOY/TEST

* Deploy the VSTO with selected user(s) and revise both the design and functionality as needed.

Deployment will raise many unforeseen questions, so be prepared.

Mike D. will require skill acquisition in the following areas:

1. Continuing familiarity with the C# language.
2. Understanding of how to implement interfaces.

Mike’s primary concern will be designing the graphical interface of the VSTO tool and using code to manipulate worksheet data into formats compatible with the service interfaces. The code for processing imported data from the CSV, XLSX, etc. files should be as decoupled from the source as possible in order to allow reuse by other applications; this means the use of wrapper classes as often as is prudent. The goal should always be to write code that can be reused.

Noah B will be concerned with building and deploying the remaining work (defining interfaces, building SQL utilities, etc.)

1. This could easily be done, since I’ve already written classes to provide DB support for VBA. However, as you read on, you will see that this is not the main objection to using VBA. [↑](#footnote-ref-1)
2. At least, not in the same sense or with the same robustness as a ‘proper’ language like C# or VB.NET [↑](#footnote-ref-2)
3. This is not so apparent in current dicussions of a VBA-based implementation. [↑](#footnote-ref-3)