Utilizing Existing Skillsets in Business Units to Improve Application Development

January 2016

Pagos, Inc.
47 3rd St.
Cambridge, MA 02141
www.pagos.com

The conundrum of business logic

The emergence and availability of new platforms (e.g. Node.js, AngularJS, and Xamarin), open source libraries, and document-based database technologies (e.g. MongoDB) have made application development easier than ever before; yet, while these technologies aim to improve and expedite the software development lifecycle significantly, there is one facet that remains difficult as ever: the production and maintenance of business logic.

The business logic layer of the application typically requires the most developer resources and frequently proves to be the most challenging to maintain for several reasons:

> Programming business logic requires the most developer resources and usually is the most challenging to maintain

1. Developers don't typically have a clear understanding of the business processes of their company, since their primary responsibility is the production, maintenance, and enhancement of the core application.

2. The business logic layer tends to change the most frequently.

3. Often, with longstanding enterprise applications, the business logic layer resides in outdated and unsupported platforms or languages (e.g. COBOL, VB6). In these scenarios, the layer itself may have been developed many years ago and the developers originally responsible for the development effort are no longer maintaining it, making it difficult to rewrite without the risk of loss or misinterpretation of integral logic.

This complexity is amplified by the fact that business logic may vary significantly across numerous axes, such as industry, company, and region. Open source solutions are not an option for business logic itself (excluding the infrastructure of the business logic layer) due to its specificity and its proprietary nature: the business logic itself often provides the competitive advantage for each individual company in a given vertical market. It is also the most adaptive, changing frequently as the result of market conditions, regulations, and fluctuations in the competitive landscape.

Business units and limited participation in development

Let's examine the typical software development lifecycle and the associated responsibilities of each party with respects to the maintenance and development of business logic in an application.

The process begins with business analysts working with the business units to understand the processes and prepare requirements documentation. These

specifications are then delivered to the development team(s), who proceed to review the specifications, analyzing for inconsistencies, problematic approaches, additional implications of the changes, and so on. This process results in modification of the initial requirements until they are satisfactory from both the programmatic and analytical perspectives.

Subsequently, the developers translate the outlined requirements into programmatic changes in the application's codebase. These modifications are then deployed to an active environment, where they are tested by QA teams based on scenarios and test cases established in a collaborative effort between QA and the business analysts. Any bugs that are reported are then reviewed and resolved by the development team, which then require retesting by QA. Once all known issues are resolved, the final version is deployed or delivered.

By the time this process has finished, there may be additional business logic changes that need to be made. The development teams are perpetually aimed towards catching up with the requirements defined by the business units and the business units often don't receive their updates in the desired timelines, often resulting in lost market opportunities.

Actively involving the business units in development

The current approach to maintaining business logic is outdated and ineffective, as it demands constant attention and correspondence amongst all active teams in the process. An improved approach must emerge that more actively involves the business units and analysts in the production and maintenance of business logic, which can significantly cut down the time and costs associated with each development cycle. Including the business professionals more actively can also simplify change management by removing the developers from the maintenance of business logic to produce a quicker turnaround for these modifications, resulting in faster overall time-to-market.

Excel and VBA are the programming languages of the business units

Insofar as Java, C#, et al., are the tools most readily associated with the production of developers, Excel and VBA are the programming languages of the business units: business units are capable of building extremely complex business models and algorithms in Excel using built-in formulas, functions, and VBA code. Of course, there are several other business-oriented programming languages (e.g. COBOL, MatLab, and Mathematica) that are popular in enterprise-level market solutions, but the steep learning curve and relatively minimal modern user base renders these approaches less effective.

The concept of using Excel-based tools developed by business units directly in an application is not altogether new. However, these approaches typically still require specific technical expertise in particular programming languages and each application requiring consumption of the business logic has to implement these component

spreadsheets individually, making it difficult to code and manage changes around this interface.

The ideal approach involves utilizing the concept of a service-based architecture, namely microservices. Many companies are already moving towards the microservices architecture and employing different teams and programming languages to produce a modular application in opposition to the traditional monolithic approach to software development. Utilizing Excel and spreadsheet-based web services fit very well into this modern approach.

Implementing Excel models as spreadsheet-based services

The implementation is simple: the Excel model effectively becomes an interface that accepts input, processes the effects of those changes upon the model behind the scenes, and returns any requested output (I/O). Since ranges in Excel can be named, these conventions will constitute the naming for both the submitted input and the requested output in a JSON request schema to the web service. Each Excel model is accessible via the same simple I/O interface, but each model is constitutively treated as a separate service, differentiated by a unique identifier, and can be accessed independently of one another.

Turn Excel models into microservices that accepts inputs, processes the effects of those changes upon the model behind the scenes, and returns any requested output

This approach drastically changes the responsibilities of each respective team associated with the development effort. The business logic can be tested by the business units in Excel and simple, auto-generated, web-based, user interfaces (through a quick, automated 'conversion' process) can be exposed internally for subsequent testing without any developer intervention using SpreadsheetWEB. Business units can then modify and retest those services by updating the underlying spreadsheet file directly in the Excel platform and subsequently expose those alterations by re-uploading to the SpreadsheetWEB server for immediate access by other internal users.
Once the logic is fully vetted, the services are turned over to the development teams to connect the service to the actual, production-ready user interface or to the internal processing of the application. If the I/O mapping from the application to the service has not changed from the last iteration (which is most frequently the case), then nothing needs to be done from the coding perspective. If the I/O has been changed in the model, then it requires a simple mapping exercise to update the application's interface to the changes in business logic.

Lead by example

Let's take a simple example to walk through the process: estimating shipping costs. The process of calculating estimated shipping costs can be easily replicated in an Excel spreadsheet, incorporating various lookup tables, discount rules, distance and weight based calculations. This logical model can be built entirely by business experts using worksheet formulas in Excel.

Despite any of these complex calculations and table structures (signifying the business logic layer of the application), the user input section is very simple:

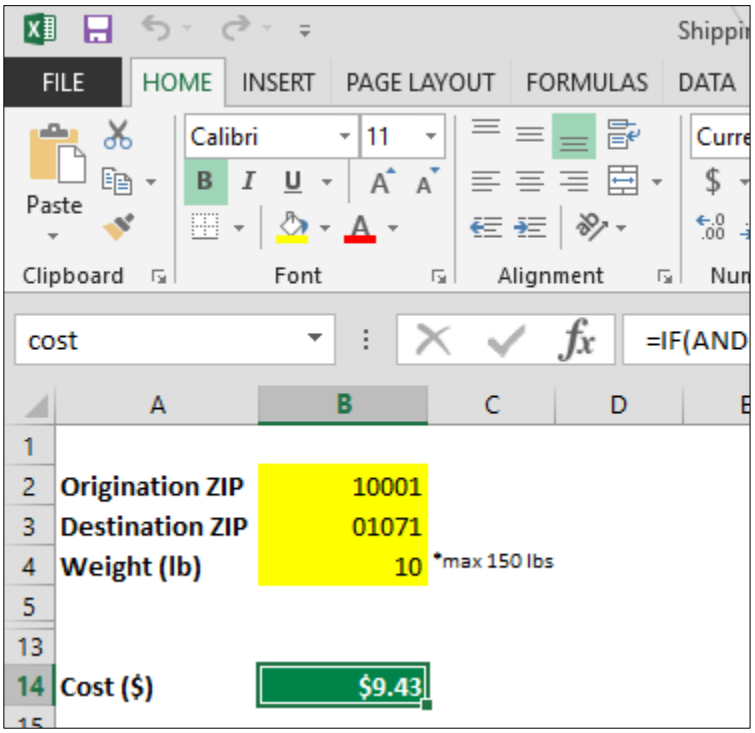

Figure 1 - 'Input' worksheet of the Shipping Calculator.

From the developer's perspective, this is the important portion of the application since it constitutes the user interface that needs to be developed in order to interface with the back-end business logic. All that the developer truly needs to know is that there are four significant fields:

Type (I/O)	Name (Identifier)	Label
Input	originZip	Origination ZIP
Input	destinationZip	Destination ZIP
Input	weight	Weight (lb)
Output	cost	Cost ($)

To convert this workbook into a web service, we simply need to upload the Excel file to the SpreadsheetWEB server and obtain an application key:

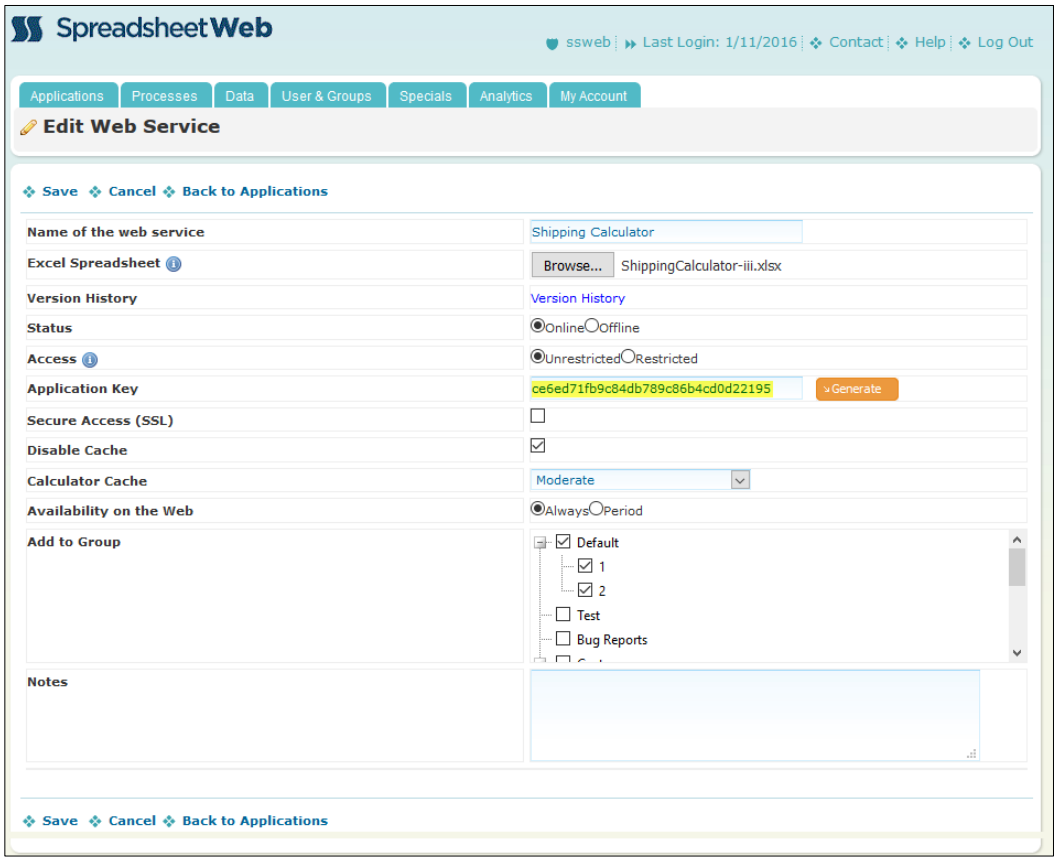

Figure 2 - Upload the Excel file and generate an application ID.

At this point, the process of submitting the user input and retrieving the output is incredibly simple, effectively a mapping exercise from the I/O of the user interface to the web service:

```
// Instantiate a new SpreadsheetWeb API class and define target endpoints.
var api = new SswebAPI(false)
{
        APIServiceURL = "http://dev7.pagos.com:2150/JSONService/",
        OAuthServiceURL = "http://dev7.pagos.com:1961/OAuthIssuer/"
};

// Define your inputs and associated values.
var inputList = new Dictionary<string, List<string>>
{
        {"originZip", new List<string>() { inputOriginZip.Text }},
        {"destinationZip", new List<string>() { inputDestinationZip.Text }},
        {"weight", new List<string>() { inputWeight.Text }}
};

// Define your requested outputs.
var outputList = new List<string> { "cost" };

// Call the API using the application key that you retrieved when uploading
// the business model and receive your results as a serialized object (JSON).
var result = api.GetResult("ce6ed71fb9c84db789c86b4cd0d22195", inputList, outputList);
```

For the purposes of this demonstration, we have created a simple WinForms application in C# to correspond with this I/O, but there are also readily-available integration libraries for interfacing with the SpreadsheetWEB API from your Java, JavaScript, and PHP applications. This web service can be consumed by any type of application, web-based or otherwise.

We now have a fully functioning application that's entirely decoupled from the underlying business logic:

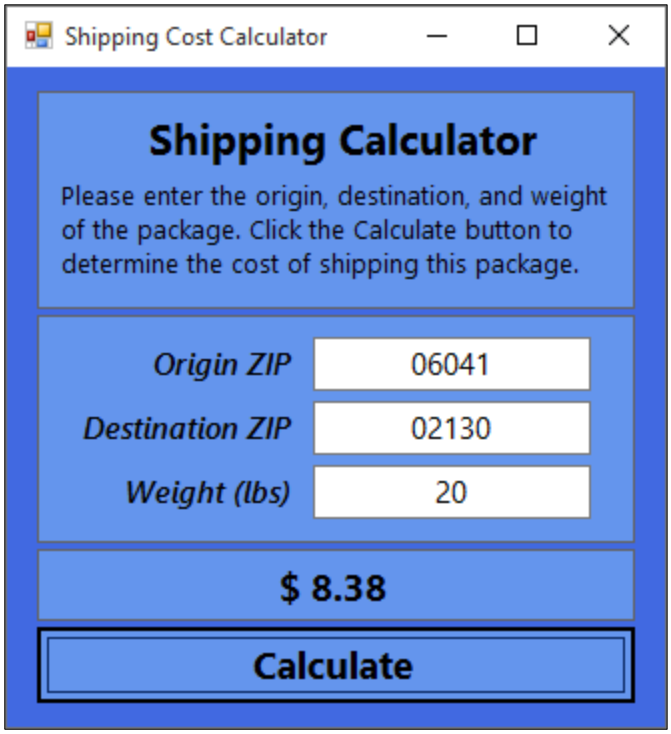

Figure 3 - Simple sample application consuming the SpreadsheetWEB API for all business logic.

As shipping rates are adjusted, the business units can freely update and test the underlying logic with new data, modified rules, and revised calculations. Then, they simply re-upload the Excel file to the SpreadsheetWEB server, replacing the existing one and – so long as the basic input and output fields are not modified – none of these business logic changes will require modification to the core application's code.

Enterprise-wide business logic services

Effectively, this approach allows for the creation of enterprise-wide business logic services, where business units control the logic in Excel and various applications connect to the related services that are all running on a single server. In a single enterprise application, various modules aimed at differing portions of the business would consume their own logical models via web service. For example, CRM software could connect to a product pricing calculator, fulfillment software could consume a shipping calculator, and accounting software could connect to a budget estimator – all accessed through the same simple interface.

Scalability can be achieved to increase handling increased volumes of user requests in SpreadsheetWEB and these web services can also be exposed externally to any registered partners by providing them with the application key and a set of security credentials: the entire security model is already built into SpreadsheetWEB.

Honing in on Existing Skillsets

In summation, the skills that are required to optimize and augment your software development lifecycle are already present in its current practice. The trick is to simply utilize these resources most effectively in order to achieve the optimum performance of every team involved in the process. With SpreadsheetWEB, you can delegate the business back to the business units, who know the logic best, while keeping your developers aimed at improvements towards the core application.

For more information, contact
Pagos, Inc.
47 3rd St.
Cambridge, MA 02141
(860) 674-9100
info@pagos.com