Knowledge Dynamics®

PERFORMANCE TECHNOLOGIES

Spreadsheet Programming:

The New Paradigm in Rapid Application Development

Contact: Info@KnowledgeDynamics.com
www.KnowledgeDynamics.com

Spreadsheet Programming: The New Paradigm in Rapid Application Development

A new software development technique known as Spreadsheet Programming dramatically improves the efficiency of developing complex software. The technique delivers extraordinary cost reductions by making programmers more productive and allowing business analysts and other non-programmers to play a role in implementing business logic. Productivity improvements of 90% or more are not unexpected.

The essence of Spreadsheet Programming is this: Corporate software development teams are using spreadsheet applications such as Microsoft Excel to build and test program logic and business rules, which are then integrated directly into production systems.

One way to implement the technique is by integrating Excel into applications using COM or Visual Basic for Applications (VBA). This approach, however, has scalability, performance, and distribution issues.

A product such as KDCalc™, from Knowledge Dynamics, Inc., facilitates Spreadsheet Programming by compiling Microsoft Excel spreadsheets into executable code that runs independent of Excel. It essentially turns Excel into a development environment for creating highly sophisticated data processing systems with multiple inputs and outputs.

This whitepaper offers an introduction and insight into the technique of Spreadsheet Programming and provides several case studies highlighting the benefits and efficiencies it delivers.

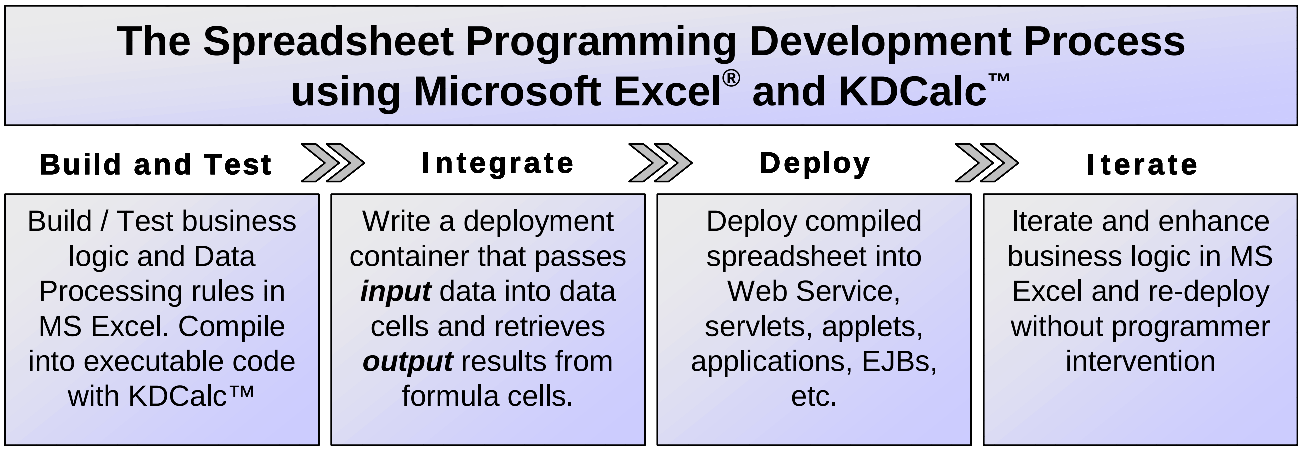
Rapid Application Development Using Spreadsheet Programming

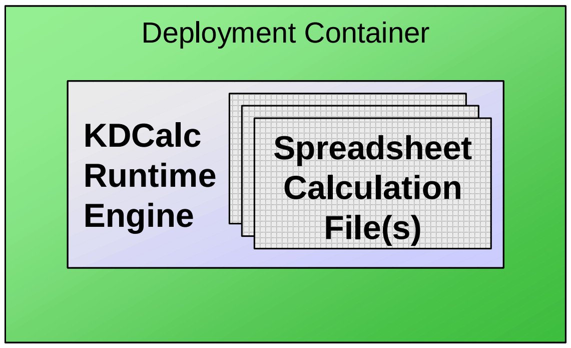
There can be little doubt that spreadsheet applications have been one of the most influential forces driving the adoption of computers for business and personal use. The ease with which ordinary people can rapidly and visually create complex logic, what-if scenarios, and simulations has transformed the business environment.

Spreadsheet Programming brings these same efficiencies to software development.

In Spreadsheet Programming, spreadsheet applications such as Microsoft Excel are used to build and test decisioning algorithms, simulations, reporting, or any other kind of sophisticated calculation and data processing system.

For single-user applications running on Windows, Excel can be integrated directly using COM or VBA. However, for server-side or web based applications, direct Excel integration is awkward or impossible. In these situations a component such as KDCalc from Knowledge Dynamics is ideal, producing small, fast, easily integrated calculation engines. **Figure 1** below illustrates the development process and possible deployment scenarios:


The Spreadsheet Programming Development Process using Microsoft Excel® and KDCalc™
Build and Test
Integrate
Deploy
Iterate
Build / Test business logic and Data Processing rules in MS Excel. Compile into executable code with KDCalc™
Write a deployment container that passes *input* data into data cells and retrieves *output* results from formula cells.
Deploy compiled spreadsheet into Web Service, servlets, applets, applications, EJBs, etc.
Iterate and enhance business logic in MS Excel and re-deploy without programmer intervention

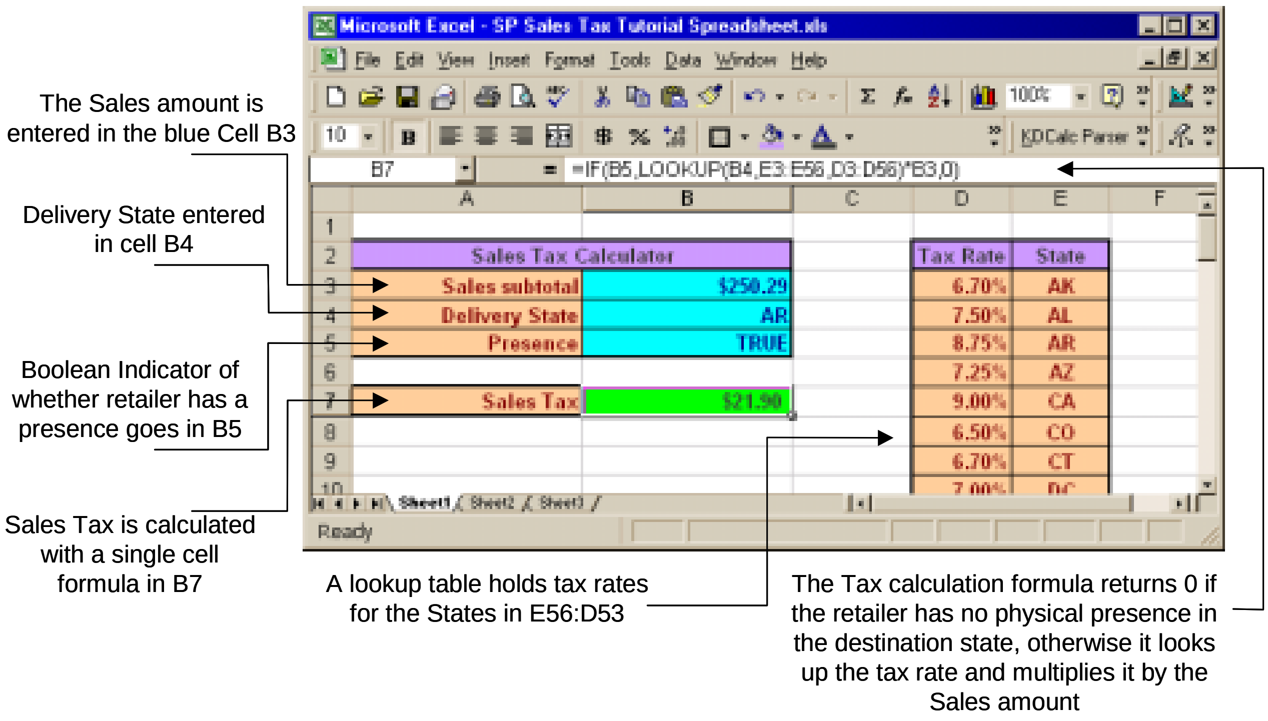


Deployment Container Can Be:
- Web Service
- Servlet, EJB, JSP
- Application, Applet, DHTML
- etc.

Figure 1: Spreadsheet Programming development process and deployment scenario

A Simple Example: Imagine that you are a developer at an Internet retailer. By law your company must calculate and collect sales tax for goods sold in any state where it has a physical presence. Imagine that as a developer you have been tasked with writing the module that calculates sales tax for Internet orders.

Figure 2 shows how the tax calculation might be developed using Spreadsheet Programming.


Microsoft Excel - SP Sales Tax Tutorial Spreadsheet.xls
File Edit View Insert Format Tools Data Window Help
KDCalc Parser
Sheet1 Sheet2 Sheet3
Ready
The Sales amount is entered in the blue Cell B3
Delivery State entered in cell B4
Boolean Indicator of whether retailer has a presence goes in B5
Sales Tax is calculated with a single cell formula in B7
A lookup table holds tax rates for the States in E56:D53
The Tax calculation formula returns 0 if the retailer has no physical presence in the destination state, otherwise it looks up the tax rate and multiplies it by the Sales amount

B7 = =IF(B5,LOOKUP(B4,E3:E56,D3:D56)*B3,0)

	A	B	C	D	E	F
1						
2	Sales Tax Calculator			Tax Rate	State	
3	Sales subtotal	$250.29		6.70%	AK	
4	Delivery State	AR		7.50%	AL	
5	Presence	TRUE		8.75%	AR	
6				7.25%	AZ	
7	Sales Tax	$21.90		9.00%	CA	
8				6.50%	CO	
9				6.70%	CT	
10				7.00%	DC	

Figure 2: Spreadsheet with Sales Tax Calculations

Now you simply pass the transaction data (input data) into the spreadsheet:

```
    // Pass the Subtotal, State, and Presence data into cells B3, B4, B5
    taxSheet.setValue(3, 2, 250.29);
    taxSheet.setValue(4, 2, “AR”);
    taxSheet.setValue(5, 2, true);
```

and retrieve the result (output data):

```
    // Get the tax from cell B7
    double tax = taxSheet.getValue(7, 2);
```

Simple enough. It seems like it would be easy to write code to do the same thing. But in reality tax calculations are many times more complicated than this. In some jurisdictions tax is assessed at the municipal level, some states don’t tax items like clothing but tax other goods, some items trigger luxury tax laws above certain price points. On top of that, not only do tax *rates* change from time to time, but tax *rules* also change. If the company sells globally, add another exponential increase in complexity. It is easy to see how this could require a spreadsheet with thousands of cell formulas. Now think about coding this by hand in Java or C++. Ouch.

With Spreadsheet Programming, the logic can all be built and tested in a Spreadsheet Application by an expert in international tax laws. The developer only needs to pass the transaction data into the right spreadsheet cells and retrieve the results from the right cells. Furthermore, once the system is built, the tax expert can update it continually without intervention from the programmer. In fact, one could make the case that this global tax calculator should be wrapped as a Web Service and offered to other retailers, creating a new revenue stream for the company.

The Spreadsheet Programming approach has many advantages.

Developing complex logic in a spreadsheet is much more efficient than hand coding in a software language. The visual nature of spreadsheets and usability features like drag-copy have been critical to their widespread adoption. When these productivity enhancers are applied to programming, the effects are no less dramatic. Developers can create calculations and data transformation routines in a spreadsheet in a fraction of the time that it takes to do the same in code. Beginning developers can be effective at coding logic that might otherwise require a very skilled resource.

Non-programmers can control business logic. In most corporate software development efforts, business people or subject matter experts are responsible for driving the business rules and requirements. Because spreadsheet applications have always been targeted at these mainstream users, business people can implement business rules in the spreadsheets themselves, rather than handing off a design document to the development team. This eliminates a major opportunity for miscommunication and diminishes the need for developers to be experts in the business domain.

Core system behavior can be verified by subject matter experts. One of the most labor-intensive aspects of software development is testing to verify that the software meets specifications. In many software development efforts, major parts of the system must be built before any of the parts can be tested. This makes it difficult to localize the origin of errors. But, when using a spreadsheet to develop the core logic of a system it is easy to create multiple sets of test data and switch between the sets to test the spreadsheet. This means that core functionality can be verified locally, before deployment and without writing any code.

Business logic can be updated and maintained without code changes. Once an initial system has been developed and tested, business people can continually enhance the functionality of the spreadsheets and redeploy them without involvement of development resources.

The application logic can be deployed to platforms where no spreadsheet application is available. Many emerging platforms such as embedded and mobile systems are limited by a lack of available of applications and development tools. A cross-platform technology like Java lets the benefits of Spreadsheet Programming be realized on emerging and proprietary platforms.

Algorithm internals are more easily hidden from end-users and secured against modification. In traditional uses of spreadsheets, there is a risk that users can discover or modify proprietary algorithms. Employing Spreadsheet Programming to package proprietary algorithms reduces this risk because they are compiled into code.

Spreadsheets can be used for a lot more than you might think. Spreadsheet applications are surprisingly flexible. Many people think about spreadsheets in the context of financial planning and basic number crunching. But with built-in functions like 'IF', 'LOOKUP', and 'OFFSET' you can build remarkably sophisticated systems that are useful in all kinds of applications and industries, from financial systems to training simulations, embedded systems, high-volume server data processing and reporting, and even games. Most people also link the idea of spreadsheets to a user-centric grid interface, completely overlooking the value of spreadsheets as calculation engines.

Case Study: Business Simulations

It's long been known that the best way to learn a new skill is through practice. Business Simulations immerse the learner in a simulated environment where she can practice the skills she needs to be successful on the job. This occurs by challenging the learner to make the same decisions she will be faced with on the job. Skill comes as the learner refines her decisions to try to achieve the desired outcomes.

The Problem

In a Business Simulation, the learner's decisions must result in realistic outcomes. Historically this has meant hand-coding complex simulation models and spending weeks or months tweaking and debugging their behavior. This makes these learning experiences difficult and expensive to produce.

The Solution

Knowledge Dynamics, Inc. uses Spreadsheet Programming to develop the underlying simulation models for Business Simulations. The learner's input decisions and the resulting outcomes correspond to cells in a spreadsheet. Using Microsoft Excel, the company develops cell-formulas that calculate the outcomes by analyzing the learner's decisions.

The spreadsheet is compiled into java byte-code using KDCalc. Then a user interface is developed to capture the learner's decisions, pass them into the compiled spreadsheet and display the resulting outcomes. The user interface may represent the inputs and outcomes as numbers, text, graphs, or any of a number of other types of knowledge representation.

The Result

Knowledge Dynamics, Inc. has used Spreadsheet Programming to reduce the cost and cycle times of developing Business Simulations by as much as 90%. The simulation models are much easier to build and test in Excel than in code, and subject matter experts can develop the models themselves, without having to teach a programmer the material first.

Case Study: Embedded Systems

Network Infrastructure companies such as Telecoms and ISPs have network hardware spread at geographically disparate locations. It is impractical to have personnel staffed near every piece of equipment, so it is controlled remotely from hub locations. This control is enabled by *Remote Network Management* hardware that handles control requests and performs protocol translations.

The Problem

A provider of network management solutions had a need to develop a new remote network management hardware product to exploit a specific opportunity in a new market. The company had to rapidly market the product in advance of competition and also had train customers and internal engineers how to use the product.

The Solution

The company used Spreadsheet Programming to build the operating system of the new product. As control requests come in, they are authenticated, authorized, dispatched, and repeated based on logic in the spreadsheet module. Protocol translations are also handled through simple LOOKUPs in the spreadsheet.

The Result

The company was able to develop the operating system for the hardware in 1/3 of the time needed to do the same for its other products.

The most interesting aspect of this application is that the manufacturer uses the same spreadsheet modules to simulate the product's behavior for training applications. The simulation operates at 100% fidelity with the physical products because they are from the same code base. This has not only made the training more effective, but also much less expensive and scalable to unlimited users. The increased availability of training and skilled engineers has resulted in more rapid uptake of the hardware products by the company's customers.

Case Study: eCommerce

An Internet telephony company offers extremely discounted international PC-to-Phone calling over the public Internet. Calls originate anywhere in the world and travel over the Internet to the company's servers. There they transfer to long-haul carriers at discounted bulk rates to the destination.

The Problem

In addition to billing the customer, the company is responsible for properly calculating and collecting tax from the customers in each jurisdiction. Developing server side systems to correctly account for usage metering, rate application, and tax computations in this globally distributed environment was a daunting task.

The Solution

Accountants developed the logic for rate application, billing, and taxation in Excel. The only custom code needed was to push call connection data into the spreadsheets and persist the output billing data into a database.

The Result

Spreadsheet Programming made a nearly insurmountable programming problem feasible for the cash-strapped startup. Moreover, as tax rates and bulk carrier rates change, they are easily updated without the need for code changes.

Case Study: Business Analytics

A provider of enterprise Benefits and HR solutions offers a customer service help-desk feature to its clients. A client's employees can call in to ask questions about the tax and growth consequences of different investment contribution decisions. The solution provider's CSRs select from over forty specialized spreadsheets to do an ad-hoc analysis of the employee's information and convey the results to the employee. Benefits experts maintain the spreadsheets and distribute updates by posting them in a common server directory.

The Problem

This process had become unwieldy, expensive, and difficult to manage, so the solution provider wanted to streamline it. The company wanted a web-centric system that could be centrally managed and administered. The initial plan was to manually convert the spreadsheets to Java code, but this would have been an expensive development effort with high recurring maintenance costs. Changes or enhancements would mean additional complex programming.

The Solution

The company used KDCalc to automatically compile the existing spreadsheets into Java byte-code. The company built intuitive user interfaces for each of the queries the client employees have, and now offers the service in a secure, self-serve web application. The company implemented workflow processes so that the benefits experts still maintain the spreadsheets in Excel and redeploy enhancements to production automatically, without support from the IT team.

The Result

The company was able to eliminate the initial cost of translating the spreadsheets to code and the ongoing cost of making updates to the system.

Case Study: eCRM

Traditionally in industrial products markets, sales professionals are responsible for a geographic region. Over time they develop personal relationships with customers, and are able to offer discounts and bundles. The best sales professionals cross-sell, putting together ad-hoc packages on the fly, considering factors such as inventory levels and customer order history.

The Problem

A mid-west wholesaler of industrial grinding wheels and cut-off blades had built up a reputation for its tailored customer service and sales approach, but was finding that the personal sales approach was limiting its ability to grow. The company wanted to invest in a web-based quoting and ordering system to increase its order-taking capacity, but still wanted to be able to offer discounts to customers based on order history, order size, and inventory status. The company could not afford a large investment in consultants to build the system and did not have the resources to complete the project in-house using a traditional development method.

The Solution

The company's sales and finance teams worked together to build a spreadsheet that calculates discounts, taxes rates, and subtotals based on inventory levels, order size, and order history. The spreadsheet also calculates line-item margins, and whole-order profit. The small IT staff designed a web user interface, and built components that populate the spreadsheet with customer order history data and inventory data from the company's database.

The Result

The company was able to automate a critical revenue process at a low cost, using mostly internal resources. The only features developed by consultants were web pages for the user interface. The sales staff became able to pursue new customers while the company still provided existing customers with personalized service.

A subsequent enhancement to the system added the ability to cross-sell other products related to the order.