Designing Specifications for Custom VFP Applications

Whil Hentzen
Hentzenwerke Corporation
735 North Water Street
Milwaukee, WI 53202-4104
Voice: 414.224.7654
Fax: 414.224.7650
CIS: 70651,2270
whil@hentzenwerke.com

Overview

In this session, you will learn how to write a complete specification for custom VFP applications.

An Argument for Fixed Price Work

Skill level of developers run the gamut from highly skilled to completely incompetent. Customers have no way to determine the difference, and, as a result, highly skilled developers can't charge proportionately more. Suppose you're ten times more productive than the guy down the road - not an unreasonable assumption. Now suppose that developer charges, say, $40 an hour. Can you charge $400 an hour? Not likely. So, given your higher skill level, how do you make more money? It follows that if you posses a higher skill level, you can deliver the same product for less money, or deliver more product for the less money, or deliver more product for the same money as the other guy.

As a result, if you quote applications with a fixed price, you can potentially make more money per unit of time than if you bill hourly.

Doing work on a fixed price basis requires three components. First, a detailed specification that defines exactly what will be delivered. Second, a mechanism for determining the cost for producing the application. And third, a set of tools for efficiently producing it.

This session covers the first component - how to produce a detailed specification that will define what will be delivered, and how to work with the customer to make it. The second session will show you how to determine your cost for the application that you have designed. The rest of the sessions at FoxTeach will provide you with the third component - the set of tools for efficiently producing it.

Preparing the Customer for the Specification Process

Some customers are reluctant to pay for the design of an application, arguing that (1) the guy down the road will do it for free, or (2) that the design is part of the sales process and thus the cost should be born by the vendor. How do you counter these arguments?

First, the relationship with a customer begins when they first ask for work to be performed. At this point, it's time to define the terms under which you will work. This typically can be done with an Engagement Letter that defines the process of developing a specification, the charges that the customer will incur, and describes what the specification will cover. At this point, the customer can decide whether they want to continue or not.

By describing the contents of a specification to the customer at this point, they will learn that what the guy down the road means by specification - three sheets of paper thrown together while watching Jay Leno the night before - and your specification - a complete design document that is sufficiently detailed to allow a programmer to produce and a tester to test with minimal follow-up questioning.

An analogy to blueprints for buildings or machinery is also useful in explaining why the specification carries a price and why it's not a trivial process.

Components of a Specification

Cover Letter

Description

Price

Delivery Timeframe

Explanation of Fixed Price

Terms and Conditions

Acceptance

Executive Overview

General Description

Functionality

Definitions and Processes

General Interface Notes

Code Maintenance

Maintenance Screens

Buttons and Toolbars

List boxes

Pick lists

Mover Boxes

Notes button

Specific Interface Notes

Specific Interface Notes

Functional Specifications

Log On

Application Launcher

Main Menu

Menu Structure

File

Edit

Operations

Reports

Tools

Help

Description of a Typical Screen

Purpose

Access

Usage

Screen Objects

Rules

Description of a Typical Process

Purpose

Access

Usage

File Formats

Rules

Description of a Typical Report

Purpose

Detail Entity

Filter

Order/Group

Fields/Objects

Calculated Fields

Additional Notes

General Report Disclaimer

Typical Utilities

User Preferences

Data Sets

Password Maintenance

User Maintenance

Data Maintenance

System Maintenance

Technical Specifications

Environment

Operating System

Hardware Requirements

Third Party Software

Interaction with Environment

Directory Structure

File Structures

Table Summary

Original Data

Data Set Size and Throughput Analysis

Implementation

Test Methodology

Test Plan

Test Data Set Requirements

Deliverables

Training

Installation

Milestones and Delivery Schedule

Modifications

Error Handling

Application Feedback

Costing Custom Software - Gathering and Using Metrics for Improved Accuracy

Whil Hentzen
Hentzenwerke Corporation
735 North Water Street
Milwaukee, WI 53202-4104
Voice: 414.224.7654
Fax: 414.224.7650
CIS: 70651,2270
whil@hentzenwerke.com

Overview

Determining the cost of a software application has been the bane of all but a few developers. In this session, you will learn how to determine your cost to develop a custom database application. Attendance at the first session or a familiarity with functional specifications is highly recommended.

The Difference between Cost and Price

It's critical to realize that the cost for producing an application isn't necessarily the price. In fact, it probably shouldn't be, else, you won't make a profit, and most businesses are based on a profit motive.

The goal of determining your cost is two fold: first, you don't want to price your work below your cost, and, second, as mentioned in the first session, the way to maximize your revenue, and your profits, is to quote applications on a fixed price. The higher this price is, the higher your revenues and profits will be. The key, as discussed in the first session, is that you have to be explicit about what is being delivered for that price.

It is incumbent upon you to determine the value of the application to the customer, so that you can price it accordingly. The only thing that this costing methodology will do is make sure that you don't price below your cost.

The Costing Methodology

The basic premise behind this costing methodology is to determine "how many things" are in your application, and to determine your cost for producing a "thing", and then to multiply the two numbers together.

Counting "Things"

An application can be broken down into five categories: Forms, Processes, Reports, Foundation, and Other Stuff. Each of these can be broken down further to determine the number of "things" in an application.

Because the different components of an application look and feel different, and have different degrees of difficulty of production, it is nearly impossible to assign standard costs for each part. Instead, what we do is categorize each type of component as granularly as possible, and then assign weights to each piece. Multiplying the number of components times the weight of a specific component results in a number that relates to the total size or scope of the application.

The components are based on a unit that we call and Action Point. For example, a label on a form would be worth one Action Point, while a validation for a check box might be worth three Action Points. If the form had five labels and two check boxes with validation, the total number of Action Points would be 1*5+2*3=11.

In this way, two completely different applications can be compared in terms of size (and, thus, cost) by counting the Action Points.

Forms

The types of "things" that can be found on a form can be categorized into five areas. First, there are the "dumb" things, such as labels, images and other "view only" projects. The second group of things includes controls that map to a field in a table, and include test boxes, check boxes, option groups, and so on. The third group includes complex objects like combo boxes, list boxes, grids, etc.

The fourth group of "things" are non-visual - the underlying rules behind controls, such as validation, and behind the entire form, such as form level rules, triggers, and so on. The final group of "things" is a set of weights for the form itself - what type of form is it (a simple maintenance form receives a lower weight than a complex form set) - and other environmental considerations such as user security and operating system requirements.

Processes

A process is an operation that runs without user intervention, and thus does not require an interface. Some process may require a form in order for the user to provide parameters to control the process, but once initiated, the process generally needs no further interaction.

Processes are tricky - they may seem like one of those "none of the above" types of categories. However, we've found that we can generally break a process down into the following operations: (1) match two records in a table, (2) look up a value in another table, (3) assign a value, (4) insert a record, (5) create or delete a table, and (6) write an exception

Reports

A report is any type of output requested by the user - be it a printed list or output to be merged with a word processor.

The types of "things" found on a report map to those on a form. First are the dumb objects like labels and boxes. Next are straightforward output from a table - fields. We create a denormalized cursor that is sent to a report form and so the relationship of fields in the cursor to output objects on the report is generally one-to-one. The third type of thing are calculated fields and expressions - including subtotals, totals, variables, and so on. The fourth type of thing are orders and grouping levels.

Finally, since we invariably use Foxfire! for reporting, we also count how many elements are in each of the metadata tables - data items and joins - that we have set up. The more of this that we have to do, generally the less work is needed in the actual report set up, so it evens out.

Foundation

At times, you will be putting together pieces that are going to go into you foundation. This includes routines or functions that your foundation already contains, or that you are going to use to extend your foundation. How do you account for these? The answer is that you determine the number of Action Point just like any other Form, Process or Report, but then provide a weight or factor that may discount the tool so that you can spread the cost out among several applications.

Other Stuff

There will be those instances where a component simply doesn't map to one of these predefined categories. In this case, instead of just guessing randomly (remember, that's a bad thing), you can still break the component into smaller

pieces, and then make some sort of guess at how many “things” are in each of these pieces.

An example would be an OLE Automation process. Instead of just guessing “Well, I think that will take about two days” you can break out the module into functionality and interfaces, and further identify pieces of the interface like done with the Processes earlier.

Determining Your Production Cost

Now that we've got a count of Action Points for an application, we simply need to multiply it by the cost per Action Point and we've got the cost of the application. So how do we determine the cost per Action Point?

If you don't like the answer to this one very much, you're not alone. Most people don't. The answer is that you use your history of what it has cost you in the past - and most people don't have those records in sufficient detail. What we've done is take our time records - details of how long we've spent on each component of an application - and then analyzed, in retrospect, how many Action Points were in each application.

From these numbers, we've been able to empirically determine our cost per Action Point.

What do you do if you don't have a history already? The best time to start tracking these costs is now. We track time down to a fairly granular level. We break the work we do into four levels: Customers, Projects, Modules and Tasks. A Project is a unit of work that requires a separate PO or, if the customer doesn't require PO numbers, is broken our for purposes of separate costing by the customer.

A Module is a component of a Project that is a distinct deliverable. For example, a Project may consist of two sets of screens and a reporting section. These may make up three separate Modules - one can be delivered and signed off before another is finished.

A Task is one of those things - Forms, Process, Reports - that can be costed out by itself. We track time against Tasks, and then iterate after completion to tweak the weights we use and make them more accurate.