Sizing and Costing of Custom Database Software

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

Overview

The sizing and costing of custom software is still a practice that borders on black magic. However, there are scientific techniques that can provide for accurate sizing; the best known of which is Function Point Analysis. However, one disadvantage of FPA is that it is a relatively sophisticated, and thus, expensive process. Accordingly, it may not be appropriate for many applications that VFP is used for.

We'll investigate the concept behind FPA and then introduction the Action Point Counting process, an alternative method for sizing a typical database application. We'll walk through an actual 1,000 hour application to demonstrate the techniques used in this methodology.

However, sizing an application is only half the battle. The other half is determining it's financial value. We'll discuss the metrics needed in order to translate an accurate sizing of an application into it's cost, how to collect those metrics, and how to map the collected metrics into an answer to the question" "How much is this going to cost me?"

Function Point Analysis

Despite what some other companies will have you believe, IBM is the largest software company in the world. In the late 70's they were having great difficulty, as was everyone else, trying to get a handle on how big software is. There initial problem was that they were writing software for most every platform that man had ever conceived of, and it was getting rather difficult to continue to do so. In order to get a better handle on their software development efforts, they developed a methodology called "Function Point Analysis" which provided a method to size software across languages and applications.

It is a synthetic measurement, meaning it is not specific to a single language or family of languages. Thus, an application written in COBOL could be compared, using its function point count, to a completely different system written in C++. Much like the measure of “Square Feet” can be used to compare a doghouse, a beach front condo, and an office building, an accounting module written in RPG for use on a System/32 (yes, I’m showing my age, remembering a machine like that) could be compared to a process control system written in assembler for an injection molding machine. Unlike most attempts at measuring software, such as lines of code, function points gave an independent yardstick of size, and thus, cost and time requirements. So when we say one system is 500 function points, another is 800 function points, we don't have to know what language they are written in, what platform it's running on, we just know that the one is about 1.5 times the size of the other. Therefore, obviously should take about 1.5 times longer and cost 1.5 times as much to develop.

Finding Function Points

A function point is a unit of functionality delivered to the end user. How do we find function points? Well, the point of this chapter is not to go into great detail on function points, but simply some background on what function points are because we are going to take this concept and twist and bend it and mush it until it's barely recognizable. Essentially, in order to arrive at the number of function points in a system, one counts the number of inputs, the number of outputs, number of inquiries, the number of internal and external logical files as well as evaluating approximately a dozen general system characteristics to determine the overall system complexity. General system characteristics (GSC's) are things like data communications, distributed processing performance, heavy use configuration transaction rates, on-line data entry designed for end user efficiency, on-line updates, complex processing, installation ease, operational ease, and multiple sites.

After we have finished counting, we mash the counts through a formula, and end up with a final number of function points. How is that translated into dollars?

Here's the "Oh no!" part of this explanation. Just as with anything else that you're going to project, you need some history first! In other words, a historical baseline of costs for projects is used to determine the costs for future projects. If a project in the past had 300 function points and cost $300,000, then the cost per function point is $1000. Of course, there are a number of factors that tie into the cost per function point. Just because an application has a certain number of function points doesn't mean that the application will cost the same to build regardless of the language and other factors.

You can think of this as similar to the square foot measurement. We can compare two structures with similar numbers of square feet, but if one is to be built over a hilly swamp with marble floors while the other is to built on level ground with pine floors, the costs per square foot are going to be different. Similarly, to be able to do some sort of screen design in a tool like C++ may take a fairly long time because it's a low level language - and it might actually take even longer to do it in something like assembler - whereas if you were using a language like Visual Basic, it would be very easy - and fast - to put that screen together. So you also have a multiplier based on the tool you are going to use. But, again, that is strictly to determine the cost. The number of function points in a system are the same regardless of what tool you are going to use to build it.

So now we have a number of function points, we have a cost per function point that depends on our history as well as the type of the tool we are going to use to build it, viola! we have a cost!

Now the "most excellent" thing about function point analysis is that this provides a mechanism to price the work according to what the end user sees - what they are buying - functionality! If they decide to add four more screens, we simply recount the function points, run it through the formula and this is the new price. Just as if you were going through the line at the grocery store and said "Hey, I need two dozen more eggs!" You put those in your basket, you count those along with whatever else you bought, and you have a new final price. The user has the determination of what they are going to buy. If they ultimate price is so much more than what they are prepared to spend, then let's take some stuff out of the grocery basket. Well, I guess we don't actually need this second box of Captain Crunch Crunchberries, do we? So the user has control over the price because they control how much stuff they are going to buy.

There is a published standard for Function Point counting, and there are a number of companies that do Function Point Analysis consulting and will provide an independent count of function points in an application. Thus, it isn't a matter of the client having to take the developer's word "Uh, yeah, there are 1156 function points in there somewhere, but you wouldn't understand the formula, so you'll just have to believe me."

The customer and the developer can hire separate organizations to do counts and measure and come up with numbers - if the final results vary (it's a science, but it's not an *exact* science!), generally the contract between the client and the developer has a mechanism for resolution.

What's Wrong with FPA?

So why don't we use Function Point Analysis at my shop? Because, in the immortal words of Andy Griebel, "That would be too hard." FPA is a complex methodology - RDI Software is a multi-million dollar development shop - and IBM is even larger. If you're writing $10,000, $25,000 and $50,000 applications, FPA is most likely to be overkill. If, on the other hand, you're involved in a 4 man-year project that's going to span 2,000 users and three continents, then our method is probably not going to be robust enough. (Of course, if you have no methodology in place, then our method is absolutely better than that!)

Simply stated, Function Point Analysis isn't going to do you much good for a project that's going to take six weeks, but you need some sort of method to price it - because a six week application that ends up taking four months could put you under the table - forever. While FPA works well for a project of at least a certain size, we're not in that league (yet!)

Our Alternative: Action Point Counting

What I've done is taken the ideas out of Function Point Analysis and modified them to suit my company's needs as a smaller developer. What do I call this? I've jokingly called it "Function Point Lite" in the past but I don't actually want to call it that because I don't want the Function Point people or Miller Brewing to get mad at me. So I've been calling them Action Points because it's rooted in the same basic concept but the implementation and process differ.

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.

Determining The Cost Per Action Point

The next question - now that we know "how many things" (Action Points) are in the application - is how much this is going to cost? The answer is rather unpleasant for many people, because it requires a piece of information that they don't have - their history. Software developers, as an industry, are remarkably immature in this respect, in that we really don't have any idea of what our historical costs have been. I can think of a dozen good-sized MIS shops that do not collect any type of data on how long it took to develop applications in-house. These are the same companies that have an army of the green-eyeshade folk counting every penny that goes into the particular brand of widget or thingamabob that the company manufactures. And at the end of the year, the VP of MIS throws up his hands and wonders why all these software projects are over budget and late.

Why? I'll tell you why - because their estimating techniques are all built on quicksand - the quicksand of having no historical records of what they did yesterday.

Tracking Hours

You must track your costs - the amount of time it took to develop projects in the past - if you are to have any hope of assigning a time value on an action point. We've been fortunate in that, before we had any inkling of what I would do with the data, we've kept detailed records on the time we spent on various components of each project we've done over the past three years or so.

As a result, when we came up with this idea of counting action points, we were able to get realistic values from the work we had done in the past. We took about a dozen applications, totaling about 4000 hours of work, and counted the action points in each one. We then correlated the actual amount of time spent (not the time we estimated or quoted at the time!) to the number of action points, and came out with some rough multipliers.

Charting Action Points by Developer

Note that the multipliers also related to the developer who worked on the project - and if more than one developer worked on the same project, we tracked the Action Points delivered by each developer separately.

We ended up with a chart that looked like this (numbers shown are totals for each developer across multiple applications):

Developer	Hours	Action Pts	AP/Hr	Rate/Hr	
	A	1000	2900	2.9	$17
	B	1800	3000	1.6	$10
	C	2400	9100	3.4	$21

Given that Developer B is the least productive - whether that be due to experience or skill level - their rate per hour is pegged at an initial number that is used as the baseline for determining the hourly rates for the other developers. If their performance of 1.6 Action Points per hour corresponds to, say, $10 an hour (yes, I'm just making up numbers here), then the developer who can product 3.2 Action Points per hour should be charged out at $20 an hour. In practice, the disparity between production rates is higher, but you get the idea.

By the way, we also took these multipliers and went back to each individual project - the variance in most of the projects was within a close enough range,

taking into account the skill level of the developer improving and so on, so that we felt the numbers were reasonable enough to use.

Determining the Cost per Action Point

From these numbers, we now know that an Action Point costs about $6.60 (another made up number.) We just took the base hourly rate and divided it by the number of Action Points that a base developer could produce. Since the hourly rates of the other developers are relative to the base hourly rate and their Action Point multiple, the cost per Action Point is the same regardless.

Handling Variances in Developer Productivity

You might be asking right about now "So, Whil, you're saying that if a screen has 50 Action Points, and you've got a multiplier of 1.5 hours per Action Point, it should take 75 hours? But doesn't this depend on the developer as well?"

Well, there's a bit of a jump in this question. Remember that each developer has a different productivity measure - a skilled developer should be able to do this screen in less time than a rookie developer because they can crank out more Action Points per hour. Note, however, that this skilled developer is also more expensive, right? So their cost - their compensation - would be higher per unit of time. So the cost of the screen is the same - whether you've got a highly skilled developer at $200/hour or a rookie developer who can only crank out a quarter of the Action Points of the skilled developer, but charges out at $50/hour. In fact, since the goal is to determine the cost for the action point, this works out perfectly.

Additional Benefits to Action Point Counting

As you are undoubtedly aware, a highly skilled developer can easily be ten times as productive as an ordinary developer, right? In fact, to say that an extremely skilled developer could be 20 times more productive than a hack isn't a stretch, isn't it? But even the most awful hack (and we have some of them in our business, don't we?) can get away with charging, oh, $40 or $50 an hour.

But using Action Point Counting, we can take advantage of the high productivity of a skilled developer and charge appropriately.

The second benefit is scheduling. Since we have the number of Action Points for an application and the productivity of a developer in terms of Action Points per hour, it's easy to determine how much time a given developer will need to complete that application.

Don't Forget Overhead

Through our historical records, we have determined how long it took a developer to write an application, and we know the cost of that developer - their wages.

We also know how much we spend on running the firm - rent, non-billable personnel, magazine subscriptions, goofy screensavers, and so on. The difference between the billable rate and the wages of the developer need to support that amount we spend on running the firm. If it doesn't - what do you do? You need to find extra revenues to pay those bills.

The first possibility is to raise the billable rates of the developers so that the numbers come into line. The other possibility is to raise revenues by charging more than the cost of the application represented by the direct labor (the developers.)

Manufacturers have been doing this for two hundred years - including the cost of running the firm - overhead (some firms refer to it as "burden") - as part of the price. Remember, we're producing a product - a widget - and good old cost accounting requires that you include both fixed and variable costs. As your firm gets bigger, you'll need to include things like brand new Pentium Pros (won't that reference seem dated in a year or so!), subscriptions to Dr. Dobbs, Software Development, and Object mag, a fancy phone system with voice mail instead of a $19 answering machine, administrative support and all that free pop.

Where To Go From Here

Is this all too much to do in one step? Probably. If it's all new to you, the first thing to do give this "Action Point Counting" a whirl on a few apps, just to determine the raw size of the systems. The numbers you come up with aren't that important - it doesn't matter if your app is 200 Action Points and someone else's is 380. What matters is that you can compare the relative sizes of the various projects you've done, look at the actual time you spent producing them, and then get an idea of the dollars involved.

From this, you can try to price an application - or even just a single screen or report - based on the hard data you've got, instead of yet another SWAG.

Worry about the sophisticated cost accounting once you've got the basics down first.

<end>