Column: From Editor
Figures: NONE
File for Companion Disk: None

Auditing an Existing Application

While Huntsmen

It's going to be trouble. You're at a meeting with a potential customer and things are going swimmingly. They want you to convert an old application they've had running since the middle-DOS years to a more modern interface, and, while you're at it, fix a few bugs, finish up that one set of reports that the previous developer never seemed to get around to, and add a few major bells and whistles. You're pretty comfortable with them; they're making you feel like part of the family, and the budget and timeframe don't seem to be major issues. And then the quiet guy near the end of the conference room table speaks up, and says, "By the way, we'd like you to take a look at the old version. Would you mind?"

(And if you're a corporate developer who puts together apps for other areas of your company, the situation would be for a department head asking you to take a look at an app that your MIS group has never known about before.)

Wow.

This is one of those moments akin to when your girlfriend for the last 14 months asks in a seemingly innocent manner, "What are you thinking?" Although you may not be aware of it at the time, the next few moments could be the opportunity of a lifetime. Or it might end up being a deal-killer. It just depends on (1) whether you're aware of how important the next few minutes are going to be, and (2) whether you're prepared to deal with this situation properly.

Here are the pressure points:

1. How do you appear impartial?
2. How do you provide value to them?
3. How do you make use of the time you're going to spend?

How do you appear impartial?

This is tricky - and it's really important. It's tricky because you don't necessarily know the history of the app - like… who wrote it! Wouldn't you love to start ripping on an app that's obviously a total hack job, only to find out that the guy running the meeting wrote it as his first job as department head several years ago, or, worse, his nephew. None of us has a mouth big enough for that foot, eh? It's important because you don't want to appear self-promoting: "Oh, they use a 'Quit' button in the app? That's bad! Very bad! You really want to have a 'Cancel' button instead - and we've been using 'Cancel' buttons for years!"

So, you appear impartial by using an existing, pre-printed checklist.

There's a hidden benefit to this approach. It's quite possible that the customer will be having more than one developer look at the app - just as they might be asking more than one developer to quote on it's replacement.

In either case, you're going to come out ahead. If you're the first one to audit, you've laid down the rules - you've set the expectations for others who follow. Wouldn't you hate to be another developer who comes in and is asked by the customer, "So where's YOUR checklist?" <g>

And if you're coming in on the heels of others, and you pull out your list of things to look at, isn't it going to be rewarding to hear the customer mutter to themselves, "Hmph! The other guys just did it off the top of their head!"

What's on the checklist?

Even a sample checklist can run several pages; so instead of reprinting it in it's entirety here, it's included in this months Subscriber Downloads. But it's worth the time to review the major points.

Documentation. Yeah, go ahead, roll your eyes. When was the last time you saw a home-grown app with documentation? But you'd better ask. How are you going to evaluate the rest of the application if you don't have a definition of what it was supposed to do? And, by asking, you lead your user to believe that applications you build will have documentation. (And they will, won't they? <g>)

Environment. Applications, unfortunately, have to run on hardware (blech!) and interact with other software. Here's where you look at what the environment is. You'll want to look at the network, the server, the workstations, and, if applicable, how remote access is handled.
Application Structure. It used to be that a programmer could get away with slamming twenty or thirty PRG files and a half-dozen DBFs in the same directory. Not any more. An app I've been working on recently has over 10 MB of source code alone. But this area addresses more than just separating programs and data. How is installation handled? Is there test data built into the system? How are versions handled? Is the directory structure clean and well organized?
Generic Application Features. These days, most applications are built on a framework of some sort - whether it was homegrown or commercially produced. One immediate advantage is that a framework has a lot of "goodies" that just come along for the ride - lookup table maintenance, data indexing, packing and repair, a wizard for creating minor entity maintenance screens, user handling and security, and more. But if the application you're looking at doesn't have a foundation at it's heart, you'll want to see what features are there, or are missing.

Not only should specific functions be available, you should be looking for features that should be in every application. These include multi-user capabilities, help, standard user interfaces, and general friendliness.
Specific Application Features. Do all of the menu options, screens and reports work? Or are there multitudes of "Under construction" messageboxes in an application that hasn't been touched since 1995? If major functions haven't been finished, I'd bet there are pieces inside working modules that aren't done either.
Data Issues. You're looking to determine if the programmer thought out their data structures in advance, and whether or not they kept them up to snuff. Is their data clean and organized? Is it normalized? Is there a lot of trash laying around? Are the first fifteen records in each table filled with deleted records containing "Daffy Duck" and multiple versions of the programmers' friends' names? I've seen more than one application where the primary key in a table is defined as character, and the foreign keys that map to that PK are defined as numeric in other tables. Bet that programmer didn't have any bugs in their code, eh?
Programming Techniques. Here's where you finally get to where most people think an audit begins. Essentially, you're asking the question - do they have a command and mastery of the language and tools used to build the application, and, if it appears they do, did they use them properly? In the days of FoxPro 2.6, you'd inspect an application for hand-coding screens and DO WHILE !EOF() structures. Today, you'd look for applications that use nothing but the FoxPro base classes, and manually hand-code the opening and closing of files in every forms' Init() and Destroy().

How do you provide value to them?
The answer to this question has often been phrased as a question to me: Do you give the customer a copy of the completed audit? The answer? Heck, yeah! Oh, I don't give them a copy of the pages I've been scribbling notes on - my handwriting is used in medical schools for teaching physicians how to write illegibly. But I take my notes back to the office, edit out the <I don't believe this!> comments, type them up, and send back the results.

By reviewing the notes I've taken and writing them up in a comprehensible fashion, I provide the customer with more than just a verbal opinion or a first-look impression that may be skewed unfairly by a specific issue. A second pass at the audit checklist takes away the heat of the moment.

How do you make use of the time you're going to spend?
One of my fundamental premises in life is that I don't like to waste time - I'm very particular about it. Therefore, since this type of situation is one that can take a lot of time without a lot of results, it's important to me to determine what's in it for me.

Having done enough audits to last a lifetime (after you've seen two or three hack jobs, you've seen them all), they're not terribly demanding in an intellectual sense. Thus, I can spend the time "listening between the lines" of what the customer is saying. You can learn a lot by paying attention to what they talk about - and what they don't mention - and determining what's really important to them. They may talk in grandiose terms about functions A, B and C, but when the meeting is over, if you haven't realized that "D" is the critical feature above all else, you're going to head down the wrong path from the start.

One last trick

It's to your advantage to not perform the audit during the same meeting. First of all, to do an audit properly can take a couple of hours - and it's expertise that you should be paid for - not performed as part of a sales call. You can make this happen by explaining that your audits (unlike those of the hack down the street) are comprehensive, covering A, B, C, D and even E. As a result, they can't be done in five minutes, any more than you could inspect a 15 room house while the car is running. You'd be happy to provide a complete audit, of course, together with written results, and it would only run a couple hundred dollars. This way, you get to determine if they're serious, or if they don't really care that much.

Whil Hentzen is editor of FoxTalk.