Hentzenwerke's
Visual FoxPro in 2013
Series

The Business Case for Moving *Some* Applications to VFP in 2013

By Whil Hentzen

So you're contemplating VFP for a new project. Or perhaps you've got a 1/2.x application that is in dire need of an upgrade, and you're thinking that VFP is the best choice. But the other players in this situation aren't all in agreement. (Oh, THERE'S a surprise!) You need to make a business case to your management or your customer.

You could search for the Web for ideas, and spend hours wading through years-old discussions about End-Of-Life panics, irrational trial balloons floated by passionate but somewhat out-of-touch developers, and crafty presentations written by developers trying to justify their push to expensive 'solutions' that are more calculated to fill their pockets with ongoing maintenance revenues than are the best choice for their customers. Or you could stand on the shoulders of those who have gone before you.

In the fall of 2012, I wrote a 4 page article outlining the business case for VFP in 2013, dispelling a few myths that are commonly trotted out when VFP is brought up, and reinforcing several benefits that we take for granted, but that others aren't aware of, have forgotten, or dismiss until they're explained. The response to that article was overwhelmingly

good, and I've received many requests for more details. Since then, I've fleshed the business case to a in-depth 20 page discussion of the pros and cons of using Visual FoxPro in 2013 and beyond.

I've explored more myths (and debunked them more thoroughly) and provided additional benefits and explained them in more detail. Then I discussed situations where VFP is not the best choice. Finally, I compared the work required to upgrade a desktop application to VFP with what would be needed to rewrite the application in another language.

Save yourself a bunch of time. Use this paper as a foundation for the business case made to your people.

1. Preface

ISBN
978-1-930919-72-3

Copyright

Revisions

History

Version	Date	Synopsis	Author
1.0.0	10/31/12	Original	WH

New version
The newest version of this document will be found at www.hentzenwerke.com.

Feedback and corrections
If you have questions, comments, or corrections about this document, please feel free to email me at 'booksales@hentzenwerke.com'. I also welcome suggestions for passages you find unclear.

References and acknowledgments
Thanks to Tamar Granor, Ted Roche, and Rick Schummer for review and suggestions.

Disclaimer

Prerequisites

This document refers to FoxPro 1.x, FoxPro 2.x for DOS, FoxPro 2.x for Windows, and Visual FoxPro 9.0 with SP2 and all hotfixes installed. These are all trademarks of Microsoft Corporation.

2. Introduction

At one point a dozen or more years ago, Microsoft estimated that there were 500,000 FoxPro developers out in the wild. That means there were, at one point, perhaps a couple million or more FoxPro 1.x and 2.x applications in use. That was a long time ago, but one of the beauties of Fox was that 'it just runs'. Throw an EXE and the FoxPro runtimes on a machine, and it was much more likely that the machine would die before the application would.

Since then, though, it's become more fashionable to dismiss FoxPro than it was to rip on Cream after their tumultuous 1968 breakup. But wait, Fox isn't quite dead yet.

As 2013 is staring us in the face, there are tens (likely hundreds) of thousands of FoxPro applications still in use. But they're getting long in tooth. REALLY long. With Windows 7 becoming the operating system of choice, and early adopters starting to take a look at Windows 8, it's decision time for those systems.

In this document, I'm going to discuss the issues facing Fox 1/2.x applications owners. I'll debunk myths the anti-VFP camp trots out to scare folks away. Next, I'll provide solid reasons that make many types of upgrading a compelling proposition. Finally, I'll offer up a detailed description of what it takes to move a Fox 1/2.x app to VFP and compare that to what it would take to move to another development environment. You can use this information to make a more informed, and hopefully (relatively) unbiased decision about when an upgrade to VFP is the best path - and when it is not. You can take this information to your boss, your management, your customer and use it as ammunition against those who use emotional or financial self-interest arguments.

3. Scary Stuff

There are a lot of “VFP is dead, what should you do?” documents out there on the Web. The purpose of each of them is to scare you into thinking that VFP is a bad choice for anything more than a birthday card list. They're wrong.

VFP still has a solid place for certain types of applications for the next ten years. It's not the cure-all for every software project out there, but it's definitely the best choice for a subset.

The purpose of this document is to scare you away from unnecessarily spending $750,000 (plus substantial ongoing maintenance costs) for an app that can be done in VFP for $250,000 (and virtually no ongoing maintenance costs.) It's also to tell you when VFP is the right tool for the job, and when it's not.

4. Background

Let's get some dates straight first. (Sources: Wikipedia for the release dates, support.microsoft.com/lifecycle for EOL.)

- FoxPro 2.6a for Windows was released in September, 1994.
- Visual FoxPro 9.0 was released in December, 2004.
- SP2 was released in October, 2007.
- Sedna (SMFT add-ons) was released in January, 2008.
- Hotfixes were released in April, 2009.
- Microsoft support ends in January, 2015.

5. Decision Time

Let's take a look at the issues forcing people's hands to make a decision about their Fox 1/2.x app.

- While FoxPro 1/2.x will run on Windows 7 (I have several 2.x customer systems - both DOS and Windows - running on my own Win 7 box), it's not the slam dunk it was on a Win 3.1 or Windows 95 box. Depending on the configuration and the amount of RAM on the Win 7 box, memory may need to be tweaked. Permissions can also be problematic, as Windows resists the installation of files helter-skelter on the C drive.
- I don't know anyone who has tested a Fox app of such vintage on a Windows 8 box. Come'on, Fox 2.6a for Windows was released in September of 1994. That's over 18 years ago. It's probably a better investment of your time and dollars to upgrade that decade old system so that it'll have decent legs for another decade.
- The interface isn't just dated, it's way past dated. Users who have gotten used to an interface that wa suddenly 'old-fashioned' when Windows 95 came out didn't really have to make any mental shifts with Windows 2000 or even WinXP. But Vista and Win7 have changed the UI again, to the point that a Fox 1/2.x app is as dated as an IBM 360 monochrome monitor. While current users may suffer a little while longer, new users (and, in the case of vertical market applications) new customers aren't really going to put up with it. It's a clear competitive disadvantage if you're trying to sell the application.
- Fox has had a 2 GB file limit for as long as I can remember (and those of you who are snickering, "so, last week, Whil?" can go sit in the corner.) While this limit is nothing new, more and more systems are finding that 15+ years of use have produced data files that are bumping into that limit, and thus are needing to resort to some behind-the-scenes slight-of-hand to keep running. And even if they can run in under 2 GB, they're getting increasingly sluggish.
- As requirements (including client-server, Web access, mobile access) become more sophisticated, they're either increasingly more difficult, or just plain impossible, to implement in 1/2.x.

Bottom line: 1/2.x systems don't have much life left in them.

6. Choices

So we've got an application that was written in the early to mid 90's, has been fine-tuned over the years, contains years ore decades of accumulated code that was likely written by a cadre of developers, some of whom never met each other, a not insignificant amount of associated cruft, a user base that has gotten used to its idiosyncrasies and nuances (and without much, if any documentation), and likely, the original programmers are long gone.

Current FoxPro users have several potential solutions to their quandary.

The first choice is to do nothing. A lot of companies have been following this path for the last decade, and are starting to worry that while 'no choice is still a choice', it's no longer a good choice.

The second is upgrading their app to the latest version of VFP. Upgrading could mean no database changes, use of a language they're familiar with (to some degree), maybe even reusing some of their code and definitely some of their business logic. The arguments against are (1) no one wants to spend money that essentially duplicates what they've already got (and "just one little change" lead down the slippery slope of scope creep), (2) VFP's EOL is looming, and (3), to some, even VFP's interface is dated, compared to the newest version of Windows.

A third choice is to rewrite their app from scratch in another language and platform. Promising when compared to #2, considering that VFP is supposed to be on its last legs. On the other hand, the investment in a complete rewrite in a new language looks staggering, and rare is the company who put money aside for this work. Beyond that, finding a new team of developers who understand the company, the business, and the current application can be a challenging proposition in many situations.

So, all in all, none of these look very appealing, so they're casting furtive glances at the box that Carol Merrill is bringing down the aisle, hoping for a last minute miracle.

7. Spreading the FUD

Fear, Uncertainty and Doubt: Gene Amdahl, a former employee and then competitor of IBM, popularized this term after he left IBM to start his own computer company to go head to head with IBM. "FUD is the fear, uncertainty, and doubt that IBM sales people instill in the minds of potential customers who might be considering Amdahl products."

I argue that since 2011, the business case for upgrading to VFP - instead of doing a complete rewrite in a different language - has actually become stronger. It will come as no surprise to you, gentle reader, that this viewpoint is anathema in many circles. Naysaysers will bring up many specious arguments about why continuing to use VFP is a bad idea. These are typically emotional arguments, using vague terms, glittering generalizations, and weasel words charged with innuendo and veiled inferences. The motivation? To sway VFP developers to another camp, and always for clear personal financial gain.

Let's look at the FUD that these folks spread.

Myth #1: The emotional argument

First, there's a very distinct emotional component in the anti-VFP crowd's mindset. It rears its head because of the personalities involved.

Think about the last time a project you worked on failed because of technology. Kind of hard to come up with one, isn't it? Not impossible, sometimes the widgets just don't fit together. Much more often, though, it's the Peopleware (thank you, Tom DeMarco) that fails, that causes a project to go south. Mismatched expectations, unrealistic goals set by unknowledgable management, ego-driven wars between programmers, these are the things that ruin projects.

Indeed, that last one - ego-driven wars - is as big a reason as any. And just as epic battles have been fought over Chevy vs Ford, Coke vs Pepsi, and the Red Sox vs the Yankees, developers wage religious wars over the choice of development tools. Fox has been in the midst of these battles since it took over the desktop from dBASE in the late '80s, and then was bought and sidelined by Microsoft in the early 90s. We're in this business because we've got the mental chops to handle the intellectual demands that a problem solving business requires. We've all been thought of as the smart ones as we've grown up, we're used to being right, having our opinion being listened to because it's the most knowledgeable one in the room. Simultaneously, developers have this stereotype of being socially inept, of having difficult with interpersonal interactions. As a result, these wars take on a particular vengeance - two socially maladjusted camps each used to always being right. (I remember watching a particularly heated argument between two peers my freshman year in college; a bystander would have thought the pair were going to come to blows. The point of friction? Not a sport team, not politics, not even the choice of a computer language. No, they were arguing which was "better", integral calculus or differential calculus. We geeks are a sorry group.)

Many supposed “technical arguments” are merely thinly disguised straw horses for emotional disdain. The fragility of the .DBF structure, the lack of a true 'data dictionary' in a supposed 'database', the overwhelming abundance of commands and functions, the lack of multiple inheritance.... you'll hear developers denigrate Fox for these and other reasons. Yet every language or development tool has its weak points; the important thing to remember is whether a weak point matters in the environment that it's being used in.

Oh really?

You need to separate the emotional arguments from the valid technical reasons, and realize that many of the most heated diatribes come from folks whose growth in social maturity hasn't kept pace with advancements in their technical wizardry. Many people will argue against a technology due to heartfelt desire, rather than rational business justifications. When they're not very nice about it, it can be difficult to keep your head about you.

Myth #2: VFP is not 'cool' anymore.

No argument here. (Some might argue that it never was, but I'll address that crowd later.) Everyone wants to play with the new shiny things. VFP is very definitely not new or shiny.

Oh really?

But... so what? The 2004 station wagon in your driveway works just as well to cart kids to school and pick up building supplies from the shop. Just because it doesn't have a cell phone dock or shiny new halogen lights doesn't mean it's unfit for use. At the end of the day, is your customer going to make more money because they've got a 'cool' inventory system, or because they've got one that does what it's supposed to?

Myth #3: The technical weakness argument

The next myth argues that VFP is, technically, long in the tooth. Its interface is old-fashioned, it doesn't work tightly with operating system, indeed, it never worked hand-in-glove with the Windows subsystems, and today, even more so. SQL access can be problematic. Getting VFP to talk to the Web required the use of a third-party toolset, and getting VFP to build mobile apps is.... better left not even attempted. There are better tools.

VFP has its roots in a programming tool developed for the purpose of tracking sports statistics so that the author could better compete in the office football pool. A combination of relative ease of use (compared to other available programming languages at the time), a built-in data engine, and an English-like syntax fueled growth and widespread adoption throughout the world.

Its simple and humble roots, though, means that VFP has never been fully integrated into the Windows ecosystem. This was never a problem when building standalone or LAN applications for the desktop, but means that sophisticated desktop, client-server, Web-based and mobile systems are difficult, if not impossible, to develop.

Oh really?

But this discussion doesn't apply to those kinds of systems - we're talking about existing systems that were written and fully deployed with functionality available in 1/2.x/early VFP versions. So the fact that VFP doesn't do mobile, for example, is irrelevant. We're not upgrading a 20 year old mobile application.

VFP is technically pretty old. So let's not use it to push state-of-the-art. There are plenty of simple desktop bread-and-butter apps that need to be upgraded and don't need those capabilities. VFP will work just fine.

Myth #4: The termination of Microsoft support

Perhaps most importantly in many people's eyes, Microsoft's termination of support in 2015 is the death knell for the product. Who in their right mind would use a programming language that has been EOL'd (End Of Life'd) by it's owner?

Oh really?

Frankly, the termination of "support" is irrelevant for the large majority of cases. FoxPro 2.x apps were EOL'd in the late 90s, but somebody evidently forgot to tell the apps. There are plenty still running fine - nearly 15 years later. Who needs "support" for VFP from Microsoft? When was the last time you called them for help with your VFP app? If you have a technical question, there are plenty of forums and huge knowledge bases full of questions and answers.

And, um, even when you did call Microsoft support with a showstopper, what happened? More often than not, you were told, at $95 per incident, that the support rep couldn't replicate it, and you'd have to work around it. Perhaps you were told that it was a known issue and you'd have to work around it until it was fixed in the release (if, indeed, it was on the 'to fix' list.) Or, maybe, the party line was that the behavior you were seeing was 'by design' and, yes, you guessed it, you'd have to work around it.

Myth #5: An update to Windows might break Visual FoxPro

I've heard the argument that an update to Windows, be it through a security patch or a mainstream update, could 'break' Visual FoxPro.

Oh really?

This argument is a classic deployment of FUD. One can come up with dozens of equally implausible, but technically possible scenarios. What are the realistic chances of this happening? I'd love to hear someone describe a reasonable scenario of how a Windows patch could break VFP. One of the beauties of VFP is that it is NOT tied into the OS like many of Microsoft's other products. Throw your EXE and the runtimes in a folder and, voila!, you're done. (This is discussed at length later.) I'm not a systems guy, but I can't imagine how a change to the OS could prevent VFP from working.

Let's assume, for a moment, that this is technically possible. Would, and could, Microsoft let this happen? VFP is used throughout corporations and small businesses worldwide; as recently as 2006, it was the 12th most popular language in use, according to the TIOBE Programming Community Index. That was a year before it was announced that VFP would not be rev'd past version 9. Many of Microsoft's most important customers depend on VFP as much as they depend on Office and .NET. While Microsoft's VFP team is essentially non-existent, the hue and cry raised by VFP users everywhere in the event of a catastrophic change to Windows that broke VFP would be heard immediately, and couldn't afford to be ignored by the Windows team.

Myth #6: VFP might be unable to meet a business requirement in a specific industry

One consulting company put forth the argument that "one of the biggest dangers would come should new data security requirements come to an industry that VFP may be unable to meet." They posited that "the time-frame allowed for meeting new requirements may be less than the time required to redevelop the program in other tools, such as .NET."

To me, this sounds like the argument is that VFP will be technically unable to meet the new security requirements, and that another language would be needed to meet those requirements.

Oh really?

This is another classic deployment of FUD. There are a lot of 'may' and 'perhaps' and 'it is possible' phrases thrown in there, but much like Myth #4, I'm having a hard time, no, I'm having an impossible time thinking up even a single scenario where a data security requirement could be imposed on an entire industry that VFP couldn't meet. I'm willing to listen, but the consulting company's argument was absent of any concrete examples.

Myth #7: "VFP developers are hard to find"

I've heard people complain that there aren't any FoxPro developers to be found any longer, that the pool of VFP developers has been shrinking; that the available top developers have left the field the fastest. This is one of those specious claims that is bandied about and argued by means of circumstantial evidence and third-hand stories. "I know a guy who has been looking for a FoxPro guy for a year and he hasn't been able to find one."

Oh really?

This is nonsense.

At the risk of sounding self-serving, go to hentzenwerke.com, write down the emails of all of the authors and editors, and email a handful. A dozen emails will find you a highly skilled and available developer. Case closed.

Yet you'll still hear people with this lament. What they often mean is "VFP developers with 10 yrs of experience who will work for $50/hr are hard to find." Yes, yes they are. A few years ago, there was a job opening in the south that was being tossed from one headhunter to the next like a hot potato - and they couldn't understand why someone wouldn't relocate for a 6 month, $45/hr job.

So, yes, CHEAP VFP developers *are* hard to find; if that's what you need, good luck with that. Maybe there's a reason that $50/hr VB, Java, and .NET developers are easy to find.

If you want to argue the point of whether the development pool is shrinking, try this. Look up a list of user groups for a competing language, be it .NET or Java or whatever, from the late 1990s. Identify the officers of those groups, and find out what language they're using now. Do the same for Visual FoxPro user groups. Compare the percentage of officers who are still around and active in their respective development communities. Don't rely on word of mouth and the rumor mills.

Myth #8: VFP may not work on future versions of Windows

Meet yet another episode of FUD, making guesses and accusations about the future. There are a lot of branches to this argument, whether Fox will run on Windows 7, or Windows 8, or beyond. And whether it will work on 64 bit boxes. I heard one firm claim that VFP has to run in 32 bit compatibility mode on a 64 bit machine, and inferred that this is a bad thing. Or those VFP programs may run slower than if they had been able to run natively.

Oh really?

I'm not sure what the argument is. Yes, VFP will run in 32 bit compatibility mode... exactly the same way that FoxPro 2.6 for Windows (a 16 bit app) has been running on 32 bit versions of Windows for the last decade.

Yes, they may be slower in compatibility mode than if they were native. Well, duh. What's ignored in this argument is that the machine they'll be running on will still be so much faster than the machine the VFP app was originally developed on and tested for use - it'll still be faster than it originally was! Ever see a FoxPro 2.6 for Windows app run on a new XP machine? It's blindingly fast, much faster than that 2.6 app ran on the 386-level processor it was originally created for.

As for VFP just not running on a future version of Windows... it does run on Windows 7 and on Windows 8 in classic mode. That takes us through 2020. Past that, well, duh. A LOT of programs may not work on future versions of Windows. In fact, it's already been shown that ALL of the current versions of Office and the various Microsoft server products won't run in the most powerful version of Windows 8.

8. Advantages of VFP

Now that we've looked at the cons, it's time to discuss the other side of the coin. VFP is not a panacea, not by a long shot. There are clearly some situations where moving away from VFP is the better choice. But it's not a universal truth. Let's look at some advantages that VFP has over the competition.

Benefit #1: VFP developers are careerists

At the risk of painting with too broad a brush, developers with other languages often get bored even though their toolset regularly gets rev'd. For example, there are ongoing discussions between .NET developers at conferences, user group meetings, and online where where they are getting bored with .NET. .NET is maturing after more than 10 years of development.

Even the .NET framework only rev'd from 4.0 to 4.5 the last time around. Additionally, Microsoft is not adding things to .NET that appear to be high on developer's ER lists. However, they had the time to revamp the UI to monochrome, which is drawing a lot of flak.

Many of the things that are "new" in .NET are simply overhauls of other parts of .NET - parts of .NET that you spent six months learning and developing expertise with. And now that expertise is suddenly old-fashioned, a bad practice, or actually obsolete or deprecated.

As a result, .NET developers are pulling up stakes and seeking out new languages.

In contrast, many Fox developers have been in place for two decades or more – how long have YOU been using FoxPro? We're happy and content using it. The best practices we learned 10 years ago are still best practices today. We've gotten GOOD at FoxPro; we are efficient and versatile. And we're not bored. Indeed, even today, don't you get a small thrill every time you fire up the IDE and start typing into the Command Window? I'm reminded of the saying, “I'll give up FoxPro when you pry it from my cold, dead fingers.”

Various .NET MVPs have been regularly looking for 'the next best thing', checking out Ruby, Python, Haskell, Scheme, Objective C, among others. To be sure, VFP MVPs regularly migrated from Fox to other languages over the last decade, but not because of boredom – rather, due to the fear that the language wasn't going to last.

Not always, though:

While there are no definitive stats on the average age of developers for various languages, it would be a fair guess that Fox developers are generally older than those in other languages. Scan the thumbnails of developers on various forums and you'll see a lot of grey hair on Fox venues, not too many hipsters with gages or young up-and-comers in Armani suits and a leather briefcase. There are not too many 25 year olds picking up Fox as their next 'go to' development tool. The careerists in place are looking forward to retiring, not buying homes and having babies.

Benefit #2: No license fees

We seems to forget that in the Windows world, VFP's model of NO LICENSE FEES is a bastard child. Once you spent your $795 for VFP, nobody in your development ecosphere needs to spend another nickel - not for server licenses, client access licenses, runtimes, nothing. While the lack of ongoing licensing revenue is one of the primary reasons for Microsoft's gradual abandonment of VFP, it's a big win for our customers. They appreciate that all they have to pay for is the development time.

In other development environments, an application has many component costs: the application server, the user access licenses, the database server, client access licenses, and perhaps licenses for other components. Upgrades have their own chain of costs as well. Rare is the VFP application that has any of these costs.

Not always, though:

While the core product is royalty and license free, a fair number of VFP add-ons come with a license fee, either per developer, per installation or per end-user. And there are VFP alternatives that are also free of license fees.

Benefit #3: VFP is stable

The product has barely been touched in the last ten years. Some may argue that this is a bad thing, but I say, au contraire, papa bear.

First, stability is excellent. We know the language, we know how it works, what the bugs are, how to work around them. We know the tricks and the tips. A recent email to me said,

Yes, there are many ways to do anything in Fox, and my way is likely not the best. But, after 20+ years of working, pretty much daily, with the Fox (since Foxbase), I don't run into many situations where I wonder "How do I do this?"

This means that our time spent at work is actually developing our applications, not learning the product, coming up to speed with new features, or figuring out workarounds to bugs that have just shown up in the newest version. And when you DO have a question, the Web has the answers. There are very rarely situations where someone runs into an implementation problem or a programming bug that is 'brand new'. Somewhere, someone has already done what you're trying to do, and wrote about it. The answers are out there.

Second, it's reliable. While not bug-free (what software is?), the codebase for VFP has been virtually the same for nearly a decade. VFP 9 was released in December of 2004; Service Pack 2 approximately three years later, and a set of hotfixes not quite two years after that. The code hasn't changed in the three and a half years since. This means that our code that worked three, five or eight years ago still works today.

Third, it's always been backwards compatible - and always will be. Slowing the adoption of Python 3.0, released in December of 2008, has been the fact that it's not backward compatible with Python 2.x versions. Comparatively, applications written now won't have to be re-engineered to handle missing components that were deprecated or removed in a previous version of VFP.

And finally, applications are not going to have problems when the next version comes out, because there won't be another version. How many times have you written an interface to an Office product, only to have it break when Office gets upgraded, because the object model was changed (or existing functionality somehow was broken.) Code that is written in VFP today won't break when "VFP XXX" comes out, it won't have to be modified to deal with the new "VFP 2016" features that broke existing functionality,

Not always, though:
It's still possible to get the dreaded "C5" error in certain situations. The Fox team spent a lot of effort in identifying the causes of those errors in the last several releases, so they're fairly rare. But they're still possible, and more likely if you're pushing Fox to the edge, such as building multiple COM servers or using a number of complex controls in concert with each other.

Benefit #4: VFP is fire and forget

In its simplest form, you throw your application's executable and the VFP runtimes into a folder, throw the data into a separate folder, and you're done. Sure, you can use an installer to do the work automatically for you, and in some situations that's a more appropriate approach. The bottom line is that your VFP-based application isn't intertwined with Windows. Updates to Windows don't break our applications the way they can with languages which are tightly bound to the operating system.

I've built perhaps a dozen medium-sized (1,000 hours or more) VFP apps in the last five years and for each one of them, I delivered the app, turned the source over to the company for future tweaks, and the game was done. My customers don't have to budget for regular upgrades just to handle the new version of a product that doesn't add any new functionality. That's a big source of comfort for many customers.

Secondly, consultants dissing VFP, promoting instead a language that is getting revved regularly, are acting in their own self-interest. They move a customer to a regularly upgraded toolset and lo-and-behold, have created years of ongoing consulting revenue streams for themselves. How many VB or SQL Server or Java or .NET apps written ten years ago are still running 'as is'? They aren't.

VFP's 'fire and forget' capability is good for customers - but not so good for developers who ship the app and don't see ongoing support revenue.

Not always, though:
Some environments require a more sophisticated installation routine, including the use of a third-party installer which puts components in various locations across the Windows folder structure. And the newest versions of Windows, with their restricted access to the C drive, makes setting up a simple Fox installation more difficult. Can't just create a "MyFoxApp" folder in the root of C, load your files in there, and be on your way. You'll need to learn to use an installation setup tool. Not a big deal, but more complicated than the way we used to do it.

Benefit #5: VFP is being extended

When "open source" became the sexy movement in software during the 2000s, cries to Microsoft were heard throughout the land They asked Microsoft, if they weren't going to rev it past 9.0, to make VFP open source so that it could be kept live by other parties. Microsoft said 'no', claiming that a large number of FoxPro customers could no longer use it as it would not be from an approved vendor.

However, Microsoft enabled VFP to be extended through a couple of mechanisms. The first was Sedna, a set of add-ons that support and extend interoperability with variety Microsoft products such as SQL Server 2005, the .NET Framework, Office 2007, and other server technologies. These add-ons were released under Microsoft's "shared source" license, and were made available on Microsoft's open source project hosting site, CodePlex.com.

Next, the community created the VFPx site on CodePlex (vfpx.codeplex.com) to host additional Visual FoxPro extension projects. As of late 2012, there are 18 projects that have been released for production use, five that have release candidates (in other words, close to being released for production use), five more in beta, and ten that are in various stages of planning or early development.

Not always, though:

VFP's core engine isn't going to change. We'll never (and, yes, I realize 'never' is a long time) have a native object-oriented menu builder, new base classes, or the ability to use DBFs larger than 2 GB in size. Web access (in whatever form you want) will always have to come from a third-party tool, and VFP on mobiles simply isn't going to happen.

Benefit #6: VFP development costs less to build

Ask any developer who has built a half dozen desktop apps in both .NET and VFP how the costs compare. The handful of long-time .NET and VFP developers I've talked to say that the same desktop app in .NET will cost anywhere from twice to as much as ten times as much. (These numbers don't take into account the cost of the tools, add-ons, licenses, and so.)

Why is VFP less? There are a couple of reasons. First, VFP was created, two decades ago, to build desktop applications. It's good at it. REALLY good at it. .NET, on the other hand, was a Web environment first, with WinForms (the desktop environment) bolted on afterward. If you're building a heavy duty Web application right now, while amazing things have been done in VFP and various third party Web toolkits over the years, there are better tools (.NET, Python, PHP, Ruby, and others) that are tuned to build those applications. But for a desktop app with 20 users, well, that's VFP's bread and butter.

Second, any VFP developer worth their business card has been using a framework of one sort or another to generated their standard CRUD (create, update, delete) applications for the last ten years. Whether it's a homegrown collection of classes and utilities or a commercial framework, it's likely been in use, for the most part untouched, since there was a '19' in the year. .NET has gone through 4 major revisions in that time span, which means a lot of rework and change. Having ten years of experience with the same framework means that a Fox developer can rapidly build a desktop application, much faster than any other environment out there.

Just how much faster is a function of the actual developer and their expertise, of course. Let's suppose a conservative multiple of three. That means the $250,000 upgrade of an existing desktop app to VFP will run $750,000 in .NET. And that's just the initial cost.

Not always, though:
As the saying goes, it's a poor craftsman who blames his tools. Similarly, it's not fair to ascribe all of the cost savings to the development tool. A skilled developer with another tool will outperform a shoddy VFP developer every time.

Benefit #7: VFP development costs less to maintain

The VFP app will be 'fire and forget' – once installed, it's basically done, except for functional enhancements, for the next ten years (the previous section explains why this is so.)

Meanwhile, the .NET app, already much more expensive out of the block, will require additional expenditures. It will continually have to be managed to handle updates to the current .NET framework installed. Furthermore, there is the very real danger of it (or at least parts of it) rendered obsolete and unusable as .NET is rev'd every few years, and features are deprecated and then abandoned. The initial $750,000 spent to upgrade doesn't include the maintenance costs for the next decade.

Not always, though:
Just because an application has been finished doesn't mean there'll never be any need for future maintenance. Bugs will be found, 'must have' features will be discovered, and other problems (the occasional damaged data file, for example) will surface. There's no question that the number of Fox developers is decreasing, through attrition, retirement and even death. As time goes on, the number of developers available for such maintenance will lessen. The question is, will the demand be greater than the supply?

9. The startling conclusion

And here's the big finish. An app written in VFP and deployed on Windows today can be expected to live through early 2020s. Why? Because businesses are moving en masse to Windows 7, and it's been demonstrated that VFP apps will run just fine on Win7.

Around my neck of the woods - the small and medium business world - the word on the street is that - by and large - businesses are moving to and staying on Windows 7. Businesses who had the good sense to stay on XP or 2000 (and that would be all of them, right? snicker) - are now moving to Windows 7 and will stay there for a decade. Mainstream support for Windows 7 ends mid-decade, the same time that VFP extended support ends, while Win 7 extended support ends in 2020. So VFP and Windows 7 can co-exist hand-in-hand for, at the very least, nearly another decade.

What about Windows 8?

Evidence so far shows that VFP will run on Windows 8's classic mode, and at the same time, no current applications, including Office, will run in the new Windows 8 UI (formerly named Metro.) Every piece of software currently in use will have to be rewritten to run there. VFP is in good company here.

But I'm going to suggest in many cases, Windows 8 is a non-starter, much like Vista was a half-dozen years ago.

The reviews have been uniformly bad; Woody Leonard's four page August 15, 2012 Infoworld review of the RTM version of windows 8, states unequivocally, "Windows 8 review: Yes, it's that bad.... Windows 8 is guaranteed to disappoint everyone." He follows up with Part 2 on August 17, and then continued two months later, saying "Most businesses have already decided they'll stick with Windows 7" in an October 25th request for "Windows 7.8".

I'm not going to argue the pros and cons of Windows 8 myself, having not used it. Rather, I'm simply arguing that the majority of your business customers are going to be sticking with Windows 7 for the foreseeable future, and beyond.

But don't take my word for it. Ask yourself – or, better yet, ask your customers – are they chomping at the bit with Windows 8? Have they been deploying beta versions on production workstations because the user demand is so high? Are they impatient to roll out Windows 8 to their entire userbase?

Or are they still using XP, and reluctantly realizing they have to bite the bullet and make a full scale switch to Windows 7, realizing that it'll be a lot of work, but at the same time, expecting that the investment to do so will carry them through to the early 2020s, just as XP carried them through the early 2010s? I dare you - query your customer base, and ask them their plans for Windows XP, Win 7, and Windows 8.

So, my argument is that - in general - they're not moving to Windows 8 and Metro. As a result, your VFP app that runs on Windows 7 has a decade or more of life in it. By 2021, who knows what the computing landscape will look like? Who knows what the business landscape will look like? The U.S. will have been through three more congresses, two presidential elections, and untold changes to the tax code, potentially affecting R&D and capital depreciation. Your kids will be out of the house (and, hopefully, those college bills will be a distant memory.) Technology wise, the Web will have changed, hardware will have changed, and there's always the possibility of the appearance of a game-changing technology that turns our world around.

The takeaway: The future game plan includes VFP for many apps.

10. When Visual FoxPro isn't the right tool

As mentioned earlier, VFP isn't a panacea. There was a time when Fox could do anything you could imagine. And people imagined a lot.

Fox in Amazing Situations

- The Christian Broadcasting Network 10 million record membership and mailing database ran on a network of 386s in the early 1990s.
- Mobil Oil installed a FoxPro-based billing system on each of its supertankers that communicated with headquarters via microwave relays while the ships were ocean-bound, also in the early 1990s.
- The JFAST transportation modeling application for the US Military, originally written in Fortran and run on IBM 360s, is an application that, when shown to Dave Fulton, Fox Software's founder in the mid 90s, prompted him to respond, "You have to show this to Bill Gates - it will knock his socks off."

- The original Surplus Direct website, written in Visual FoxPro and Web Connection, handled over a million hits a day.
- And Visual FoxPro was the engine behind the application that managed over a hundred gigabytes of engineering drawing data for the Chunnel.

When I talk about some of these projects to folks who don't know anything about them, you can literally hear their jaw drop.

But software development has become increasingly sophisticated and varied; client-server, Web, mobile development - these are all commonplace, and they are areas that Fox doesn't play as well in as other technologies.

When Fox makes sense

Fox made its name in being able to work with what were then considered to be large datasets - hundreds of thousands or even millions of records were easily handled on run-of-the-mill desktops, while other development environments struggled to keep up. An oft-repeated demo was that of Rushmore, the indexing technology that made Fox so fast. It was ported to Access, and, much to the amazement of Access users, the demo would search on and find a single zip code in a 46,000 record database instantly. The same demo was used with Fox, bringing back a similar result set in the blink of an eye, except that the database used was the 1.2 million row street address database of the entire U.S.

These days, with terabyte drives commonplace in home PCs, accessing a million record database doesn't capture anyone's interest anymore. Desktop applications are the day before yesterday's news. But that doesn't mean that Fox isn't applicable any longer.

Visual FoxPro is still well-suited for several types of applications:

- Replacing aging FoxPro 1/2.x desktop applications.
- Creating new desktop applications that don't need a Web or mobile component.
- Small client-server systems that can use the 6.01.8629.01version ODBC driver to connect to the database (this one shipped with VFP 6.0.)
- Reasonably simple Web applications.

Fast development turnaround, low cost of deployment, fire-and-forget installation, and rock-solid performance are all significant benefits.

And when Fox shouldn't be used

However, if the application in question needs significant enhancements to include access to other technologies, sophisticated Web access, or a mobile component, VFP is likely not the best choice.

- Applications that require interfacing with new technologies. By definition, new technologies can be problematic, given that they were invented after Visual FoxPro was last updated. The Sedna add-ons have helped in some cases, but even they are limited in their reach and future extensibility. Hard to build tools to communicate with technologies that haven't been invented yet, or that continue to change. Automation with Microsoft Office is a prime example; the techniques that worked well with Office 2000 occasionally needed to be tweaked for Office 2003, needed more changes for Office 2007, and have broken in substantial ways with Office 2010. And who knows what the story with Office 2012 will be?

- Sophisticated or high volume Web access from Visual FoxPro. In the past, this has been accomplished in a number of way over the years; an entire book that discussed the various possibilities was written in 1999. The Web Connection framework, while definitely not the only way, is the granddaddy of third-party tools used for using VFP to connect to the Web. It uses a small DLL running on a dedicated Web server that is called by Web requests; it in turn talks to a VFP-written executable that handles database access. While powerful (it has been used for applications that handled over a million requests per day), the combination of technologies is not commonly supported by hosting companies, and thus requires a dedicated Web Connection server, which proved to be more than many installations wanted to handle. These days, .NET, PHP, Perl, and Python are commonly supported by the large majority of Web hosts.
- Mid to high volume client-server applications, or small volume applications that require use of advanced connection capability or connections to new database servers.
- Mobile applications. Over a decade ago, I demonstrated techniques to have a Palm Pilot PDA communicate with a Visual FoxPro application. In the succeeding years, not only has Palm disappeared, but the rise of multiple new genres of mobile devices has far outpaced VFP's ability to connect with them. While, with a lot of ingenuity and hacking, you might be able to get a mobile device to connect with your VFP system, it's clearly a case of "when the only tool you have is a hammer, everything looks like a nail." Don't do it.

11. Upgrade versus Rewrite: A Comparison

Let's examine the "upgrade to VFP" versus "rewrite in another language" decision at a more granular level. Suppose you've got a FoxPro 2.6a for Windows application that is reaching EOL, and you're considering either upgrading it to VFP or rewriting version 2.0 in some other tool. Here are some of the major issues to take into consideration. I'll use the terms "current 1.0 version" to refer to the current FoxPro 2.6 version, "2.0 upgrade" to refer to the VFP 9.0 version, and "2.0 rewrite" to refer to the version rewritten in Some Other Tool (S.O.T.)

Data conversion

The current 1.0 system has an existing 1/2.x data set made up of dozens, perhaps hundreds of tables, possibly comprising thousands of fields, and it's chock full of live data. This data will need to be moved to the 2.0 system.

Fox considerations:

Pre-2.6 tables will need to be upgraded to (at the very least) 2.6 but more likely 9.0.

Fox 2.6 tables can either be left in 2.6 or upgraded to 9. Unless 9.0 database features are needed, leaving them in 2.6 is a painless and virtually immediate solution.

Moving DBFs from a legacy version to 9.0 is nearly painless, with only some possible field name conflicts and value checking.

S.O.T. considerations:

The new database structure and platform would likely be a SQL database.

S.O.T. should not attempt to access Fox 2.x DBFs. Any modern tool (.NET, Python, Java, Objective C) is going to need a library or driver to read 2.x DBFs, introducing another layer that no one wants to deal with. And no one using S.O.T. is going to want to deal with DBFs any longer.

The data conversion will not be a one-time process. First, if the current 1.0 version is still being modified during the development of the 2.0 rewrite, a conversion program must be written that will convert and import the data set regularly throughout the development of the 2.0 rewrite. This is due to possible changes in the current 1.0 data structures during the 2.0 development process. Even if the 1.0 data structures are frozen, a conversion program will need to be written in order to convert the data in an automated fashion. The data itself will need to be moved to the new data structures at the very least twice - once at the beginning of development so that testing can be done, and a second time at the end of the development process, so that the latest data set can be captured for use with the 2.0 application. Attempting to do this manually is just asking for trouble.

The new data platform likely will not have the same database table types as 2.x. As a result, code would need to be changed throughout the application, in the UI, the processes, reports, and any other components that are associated with the app.

In short, the migration process needs to be looked at as another application, to be designed, built, and tested.

The SQL database will need an administrator. For simple applications, this may be a part-time person or a consultant who comes in once in a while, but the larger the system, the more work involved on a regular basis. And this person will likely be a different person than the programmer of the 2.0 rewrite, and the cost will likely be higher as well.

The SQL database will likely have both a server license cost as well as costs for each client access license (each user.)

The SQL database will undoubtedly be upgraded at least twice in the upcoming decade, requiring maintenance and additional infrastructure work, including possible conversion of data as well as modification of the code itself.

UI - screens

1/2.x screen design surfaces contain code in a handful of snippets - a Read/Show combination, When() and Valid() clauses for objects, and so on. The screen design surface code invariably contains modules that connect directly with the underlying data structures.

Fox considerations:

VFP was designed for and is optimized for building interactive desktop applications.

The VFP form builder is a more advanced version of the already familiar Fox 2.x screen builder. It's fairly simple to use a 2.x screen as a reference to build a VFP form.

In some cases, the functionality of the current 1.0 app will be carried over 'as is'. The controls and actions in the 2.0 version will be so fundamentally similar to the users that no re-training or re-learning will be necessary. (Don't fall for the idea of using a migration tool, either the one that comes with VFP, or one supplied by another firm, to move your 2.x screens to VFP. Because they were built to handle so many obscure and one-off scenarios, the code produced is not good VFP code, and you will not want to maintain it.)

A VFP version simplifies the testing process of the new UI as it can be run concurrently with the current 1.0 version (provided it's decided to stay with Foxpro 2.6 tables.) And the process of testing - throw the new EXE in a folder and run it - couldn't be easier.

S.O.T. considerations:

Other tools either do not have a native desktop building mechanism, requiring third party libraries and add-ons (such as Python), or the desktop IDE was tacked on after the fact (such as .NET's Winforms.)

Other tools have completely different paradigms for building forms, so they'll need to be built from scratch.

The user interface controls available in another tool may not be available, or they may not work the same as they do in Fox 2.x or 9.0, resulting in new interface decisions and retraining on the part of the users.

Depending on the tool chosen, installing test builds may be considerably more involved than with Fox, increasing costs as well as requiring training and a learning curve to be undertaken.

UI - code

1/2.x code can be structured in a wide variety of ways. From a single monolithic PRG with hundreds of subroutines to hundreds of single purpose PRGs to a combination of the two that include a separate procedure file that contains common routines; I've seen all of these and many more variations. They all have one thing in common, though - user interface logic (enabling and disabling controls, populating them with or binding them with data), business logic and algorithms, and data handling are all frequently bound together in code segments.

Fox considerations:

The existing code can be understood by the VFP developer as an aid to development - it is part of the specification and helps in understanding how the current system works.

Some of the code (specifically libraries and routines, but also code snippets from screens) will be able to be re-used as is or with minimal modification.

Any code that is re-used has already been tested, and will require minimal followup testing.

S.O.T. considerations:

The existing Fox code will likely need to be 'translated' by a Fox developer for the S.O.T. developer to be useful as a reference, or will be useless.

None of the code can be transferred to the new system - all code will have to be written from scratch - and tested from scratch.

Processes

Programs that simply work with data, such as importing data from other systems, can be organized in the same way as UI-associated code, and have the same tight binding between program logic and data handling.

Fox considerations:

Processes built into "batch files" via stand-alone executables that operate on the Fox 2.x data structures may not need any work at all. They can often operate as a 'black box'.

In the event of an upgrade of the data structures to VFP 9, some tweaks to the code may be necessary to handle field name, auto-increment and/or referential integrity constraints.

S.O.T. considerations:

All batch files will need to be rewritten so that they can read/write the 2.0 database.

Rewriting of the batch files is an all or nothing approach and cannot be done incrementally or as needed.

Rewriting most likely includes a completely different approach, as SQL databases do not use the same 'record pointer, physical file' paradigm as Fox DBFs.

Reports

As with everything else in Fox, reports can be produced in a variety of ways. These range from reports that are constructed manually, one field and line at a time, and sent to an output device through individual commands to building temporary cursors with all the logic and processing performed ahead of time, simply sent through a report format file, to a combination of the two, where multiple tables are joined and processed through a report file that contains embedded logic and conditional processing for each field and report section.

Fox considerations:

Two excellent existing report writers built in VFP already exist, ensuring seamless compatibility with a VFP 9 application.

An earlier version of one of these report writers may have already been used in the current 1.0 version, meaning that existing reports may be able to be ported to the new version with little to no work.

S.O.T. considerations:

The existing Fox reports will likely need to be 'translated' by a Fox developer for the S.O.T. developer to be useful as a reference, or will be useless. Fox reports are single-pass report generators, with powerful controls for grouping, breaking, report variables, conditional

"Print When" logic, specialized headers, and the capability for additional add-ins, including an entire object model to granularly control every item on a page. No other report writers share the same model, so other tools would require a learning curve and translation of the code.

Report writer options would have to be researched. All reports would have to be created and tested again from scratch.

12. The takeaway

There are many desktop applications that don't need any major changes or new functionality. For those where the business needs haven't changed much in 20 years, moving to VFP is the quickest, easiest, least expensive and least risky route to take. Visual FoxPro, amazingly enough, is a pretty safe bet for the next decade.

13. Where to go for more help

Consulting: I have been building Visual FoxPro applications for nearly 20 years. I build solid, straight-forward systems that are easy to learn and take over. If you've been intimidated by the prospect of learning the Visual FoxPro paradigm, fear no longer - I've brought many experienced FoxPro/DOS and FoxPro for Windows 2.x programmers into the world of Visual FoxPro with little pain and much delight and rejoicing as they discover the new capabilities and possibilities of VFP.

Miscellaneous Resources: There are a wealth of VFP resources listed on www.hentzenwerke.com as well.

Books: Hentzenwerke Publishing, Inc. has the largest list of Visual FoxPro books on the planet. Click on "Your Account" at www.hentzenwerke.com to get on our Preferred Customer list. If you found this document helpful, keep an eye out for upcoming Hentzenwerke Publishing "Visual FoxPro in 2013" books as well.:

So You've Inherited a Visual FoxPro Application – Now What?
107 Tips and Techniques to Help You Survive

Moving Your Application from FoxPro 1/2.x to Visual FoxPro

The Big Book of Visual FoxPro Add-Ons

Lianja for Visual FoxPro Developers: A Case Study

Converting Your Visual FoxPro application to Python in 45 Days