

My 20 Year Career is Technical Debt or Deprecated

Deprecated should be my middle name

MAY 15, 2023



2



5



Share



Technical debt is easily the most used buzzword these days. People say, “We are moving fast on our MVP while minimizing technical debt!”. They mention technical debt in there to sound cool or something.

I just laugh because everything is technical debt, eventually.

My entire career is now technical debt, or the code has been deprecated.

If you don't believe that your entire career will also be technical debt, you might after you read this article. I will walk you through how things have changed over my 20-year career.



My 20 Year Career is
Technical Debt or Deprecated

VISIONARY
CTO

I was “Basic” at first...


My career started as a Visual Basic 6 developer. I built several different apps from 1999 to 2003. I think you can say that anything in Visual Basic 6 by today’s standard is technical debt or has been long replaced already. Long live “on error resume next!”

I spent a lot of time doing classic active server pages (ASP) development. At one time, I was also an expert at making websites work with Internet Explorer 6 and Netscape Navigator. It doesn’t mean much on the resume anymore!

Visual Basic, ASP, IE6, and Netscape are all long-forgotten technologies. As **Strong Bad** would say back then, “delorted!”

Old languages: Perl, Delphi, Fortran, FoxPro, ColdFusion

The
yea
pec
for
Do
circ
In t
rem
Rul
tou
lan
Pro



Discover more from The Visionary CTO

Thought leadership about startup and CTO life, including entrepreneurship, engineering leadership, product, and more

Over 2,000 subscribers

Subscribe

[Continue reading >](#)

[Sign in](#)

of favor over the last 20+
any of these languages,
ard to find programmers
t’s tough. In most
e old apps.
hot thing. Do you
allen out of favor, and it is
w available in other
jobs learning skills that

aren’t in demand. It is always a balance of supply vs demand! **Developers jump ship quickly** and always want the hot new thing on their resume.

What happened to ActiveX, Java Applets, Flash, and Silverlight?

Some of the first apps I made used **ActiveX** controls in Internet Explorer 6. At the time, they were required to do printing and other very insecure hacky things. PDFs were not as common back then, and printing from the browser was its own fun nightmare.

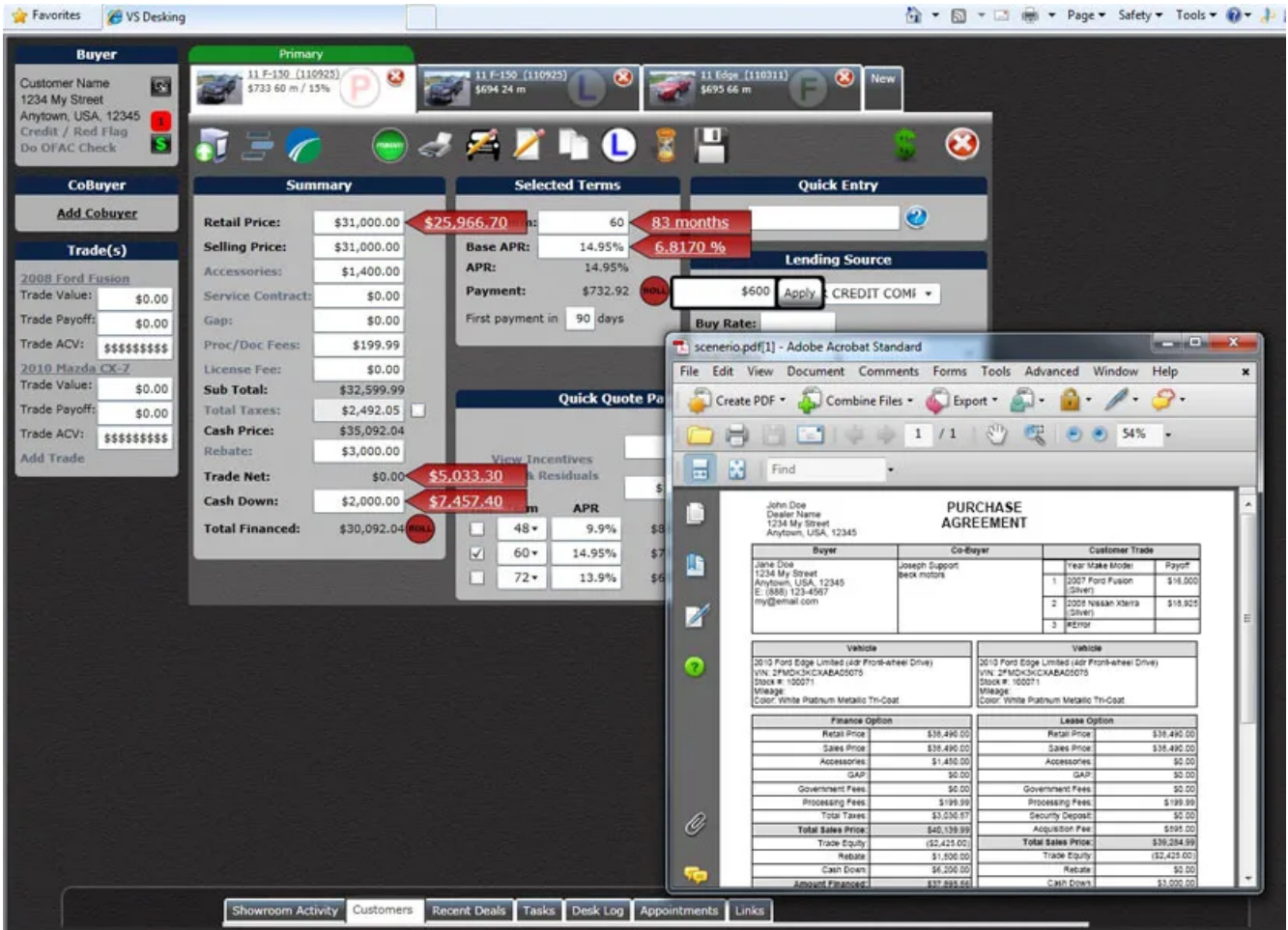
Java Applets were also a big thing once upon a time. They were slow, and having the correct version of Java installed on your computer was always a mess. I'll never forget the nightmare of dealing with networking firewalls that required Java applets. I don't miss those nightmares, and luckily it faded away.

Of course, we all remember Macromedia/Adobe Flash! At one point, it was the darling child of the entire Internet. There were endless Flash games, and lots of software was built in Flash with ActionScript. A product called CheerpX now allows running old Flash apps with WebAssembly.

Microsoft came out with a competitor to Flash called **Silverlight**. It was actually a pretty fantastic framework for C# developers. My company built some really amazing things with Silverlight.

As we all know, Apple ended Flash and Silverlight by dropping their support in their browsers.

Here is a screenshot of the finance calculator we built with Silverlight at VinSolutions over 10 years ago. Silverlight is now long gone, and they rewrote it in all JavaScript, but it is not as cool as the old version!



My first mobile app

I built a mobile app in 2004. It's hard to remember, but the iPhone and Android didn't exist then. I wrote an application for Compaq PDAs for tracking inventory for car dealers. It was written in C# for the .NET Compact Framework to run on Windows CE.

This PDA had a 1-megapixel camera. The photos were only slightly terrible as long as it was cloudy outside to remove the glare. 😂 Boy, has technology changed! This app went to the boneyard long ago but was cutting edge in 2005.



VinBuddy works with a built in camera to simplify taking photos and matching them to your inventory.

Integrated with our VinStickers™ product, VinBuddy enables the dealer to manage their inventory on and off the lot.



You better be Swift

Swift is another excellent example of how fast development tools change. As soon as Apple released Swift, it was hard to justify writing code in Objective C anymore. I am sure there are some use cases where it is still needed. But Swift is significantly easier to develop and a major evolutionary step forward.

I would argue that any apps written in Objective C are probably technical debt now.

Thank you for reading The Visionary CTO. This post is public so feel free to share it.

WebForms

After doing crazy in-line scripts for building web apps, I was happy to use the new ASP.NET web forms. Their server-side controls made development significantly easier. Their goal was to make it as easy to create web apps as you could in Visual Basic 6. It mostly worked! You could build reusable UI components server side to render to the browser. Just like we do today in 100% JavaScript.

WebForms wasn't perfect, but it was a considerable upgrade. It had a great run until Ruby on Rails came around and popularized MVC (Model-View-Controller) frameworks for developing web apps.

MVC quickly deprecated all the web forms apps we ever made. Anything in web forms is definitely technical debt. (Although, the same idea is making a comeback with [Blazor](#).)

MVC is king! (for a while)

Before you knew it, every programming language supported MVC frameworks. We switched to doing everything new in ASP.NET MVC. It was everywhere, including Django, Laravel, Symfony, Spring, etc.

Fast forward to today, and MVC has since fallen out of fashion. Everything is now done in React, Angular, Vue, and other frameworks.

Before we had those, we had other Javascript frameworks. At [Stackify](#), we used Knockout, a reasonably popular front-end framework.

Do you remember any of these frameworks? Knockout, Ember, Aurelia, Meteor, Backbone, Handlebars

If you used any of them, I bet all that code is now considered technical debt and has fallen out of favor. The first generation of front-end frameworks lost to React and Angular.

Angular JS

In 2015 Angular was created by Google and burst onto the scene. It quickly became the most popular front-end framework.

Then in 2016, Angular went through a major upgrade and was not backward compatible.

Guess what that means? Anything in the original version is now technical debt. I have projects at my company in the old version of Angular that is a major technical debt we must upgrade.

The old dirty SOAP & WCF

Before REST APIs and JSON became the de facto standard, another option was **SOAP** which stands for simple object access protocol. It made it easy to call web services and automatically code-generate proxy classes to correctly call the services. It was primarily used by the Windows Communication Framework (WCF) over XML.

It worked awesome... until it didn't. One of the worst projects of my career involved figuring out how to use security certificates between my company and another vendor over WCF and SOAP. The promise of SOAP and WCF was amazing, but it was a nightmare to maintain over time.

SOAP and WCF are two things I don't miss. Microsoft decided to no longer support WCF in .NET Core. Things like REST, gRPC, and GraphQL are now preferred. Although, a community project eventually made **CoreWCF** to keep it going.

Over time, the types of technology we have used to call web services have changed. Older ways still work, but most people would probably prefer to retire them.

Major language versions

Another common problem is major programming language version changes. Be it Ruby, PHP, .NET, or others. They commonly require a bunch of code changes or even rewrites.

When .NET Core came out, it was the newer, lighter, faster version of .NET designed to run on Linux. Basic C# code is pretty easily ported over, but nobody uses just basic code for a real-world app.

However, in complex enterprise apps, there are many potential issues when navigating the upgrade path. That becomes a major technical debt that has to be figured out.

Otherwise, you eventually get stuck on an ancient version.

Those major version updates eventually become big technical debt projects.

Stuck on an old external dependency

One of the biggest challenges we had at Stackify was getting stuck on an old version of Elasticsearch.

At one point, they made some significant changes to how it worked that were not entirely backward compatible. We used it very heavily, and all the work to upgrade became a massive amount of technical debt and upgrade project.

We kept kicking it down the curb over and over and eventually got way behind. This is another example of real technical debt projects that can plague companies.

Open source alternative retired my code

At [Stackify](#), we built our own tracing/profiling libraries for 6 programming languages. It was an incredible amount of work to do that.

Well, now [OpenTelemetry](#) has since come along and rendered all that work useless.

Why manage your own when you can use the open-source industry standard? Stackify is slowly working to eliminate the .NET profiler I helped build.

All code rots or gets replaced

Over time, you can see how almost everything you create gets scrapped and replaced for various reasons or is now based on old technology.

Several apps that I built early in my career were terminated because the companies were acquired and decided to use totally different technology.

Most software has a limited lifespan that is shorter than you think. All code eventually becomes a technical debt that everyone wants to rewrite in a more modern way, or the business needs dramatically change.

Granted, in the corporate world, it is more likely to have internal apps that seem to stay around forever. Something like a railroad or major bank has been using the same mainframe-based software for 40 years.

I predict WebAssembly will eventually overtake how front-end development is today, and a whole new world will evolve.

The reality of technical debt

People are always worried about minimizing technical debt when doing new projects. I understand that. There is a balance between getting things to work and trying to make them perfect.

However, **nothing is technical debt because it isn't perfect**. There is no such thing as perfect. Over time what was perfect today won't be perfect in the future. Learn to live with less than perfect.

The other side of technical debt is how everything slowly rots away over time. It either has significant issues with upgrading to the latest versions, or the technology ultimately falls out of favor because of newer ways to do things. Good luck hiring people for old tech stacks.

Everything eventually becomes tech debt, or the projects get sunsetted. If you are lucky, your code survives long enough to be technical debt to someone else.

Given enough time, all your code will get deleted. 🙄

Strong Bad Deleted



The Visionary CTO is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

Type your email...	Subscribe
--------------------	-----------

5 Comments



Write a comment...



John Crickett Writes Coding Challenges 6 hr ago

People are still building systems in Delphi and PERL. Delphi actually had a new release just a couple of months ago.

♡ LIKE (2) 💬 REPLY ⋮

4 replies by Matt Watson and others

4 more comments...

Substack is the home for great writing