Reflections of an 'Old Programmer' (110 comments)

October 6th 2016

Note: I was invited to speak on the Software Engineering Daily podcast on this topic, and it was a ton of fun. Check it out here.

I'm a programmer, a few months shy of his 40th birthday. It's Saturday morning, my kids are home with my wonderful wife (who is pulling my weight on the domestic front), and I'm at a tech conference. It's a session on React Native, and the presenter is convincing us why it's truly the "next big thing" for mobile development. To me, it seems a bit like JSPs of 15 years ago, with all the logic in the presentation code, but I'm "old", so I assume I just don't "get it".

The presenter blows through some slides, dazzles us with some impressive live coding, and then makes his way to the "name dropping" portion of the presentation, where he rattles off about a half a dozen supporting tools that I never knew existed, including something called Pepperoni (seriously). As someone who just recently got the hang of Angular, it all makes me feel a little disheartened. "Here we go again", I think.

Of course I'm not really surprised. Over the past 20 years, I've taken seats on a number of band wagons, and have generally enjoyed the rides. The buzz that comes with a new "disruption" in programming can be exciting - feeling apart of a community of technical innovaters, championing something that will make things a little easier, quicker, cleaner, better. It can be fun. But on this particular morning, at the cusp of 40, I have to admit I feel a little drained. I know this is part of the job - if I want to stay relevant (and well paid), I know that every so often I need to cast out some of the knowledge that I've so dutifully absorbed, and gear up up for the next journey. It's just how it is.

As I think about it though, this regular ritual of my programming career doesn't seem to be a way of life for other professionals. The doctor at 40 doesn't seem to be worried about discovering that all his knowledge of the vascular system is about to evaporate in favor of some new organizing theory. The same goes for the lawyer, the plumber, the accountant, or the english teacher. While there are certainly unappealing aspects to these professions, it's safe to say that for each of them, mid-way through their career, the knowledge they've accumulated is relatively stable, and has afforded them with some increased measure of respect and compensation. In programming though, 20 years of experience does not seem to confer those same advantages.

Two Main Forces

Of course not all is so dismal in our profession - there are so many things to love about being a programmer - but in terms of the never-ending struggle to "keep up", it is an interesting feature that seems more or less unique to our field. Am I right though? Is programming really different in this regard? And if it is, then why? And what does it mean for our career trajectory? I'd like to try to answer *all* of this (because, why not) in terms of two concepts.

The first is **knowledge decay**. Everything we know, not just about programming, has an expiration; a point at which it is no longer useful. I learned how to drive a car when I was 16, and for that most part, that knowledge still serves me well. This piece of knowledge could be said to have a long half-life. For many professionals, their domain knowledge also has a relatively long half-life. Sure, new discoveries in medicine may displace some existing procedures, but likely there will not be a major overhaul in our understanding of our biology. When the expiration is long like this, knowledge can effectively be considered cumulative. The doctor is *more* knowledgeable than he was last year, because everything he learned in the past 12 months built on all that he knew before.

In programming, for good or bad, I'd assert that this is not exactly the case. Putting a (rather arbitrary) stake in the ground, I'd say that:

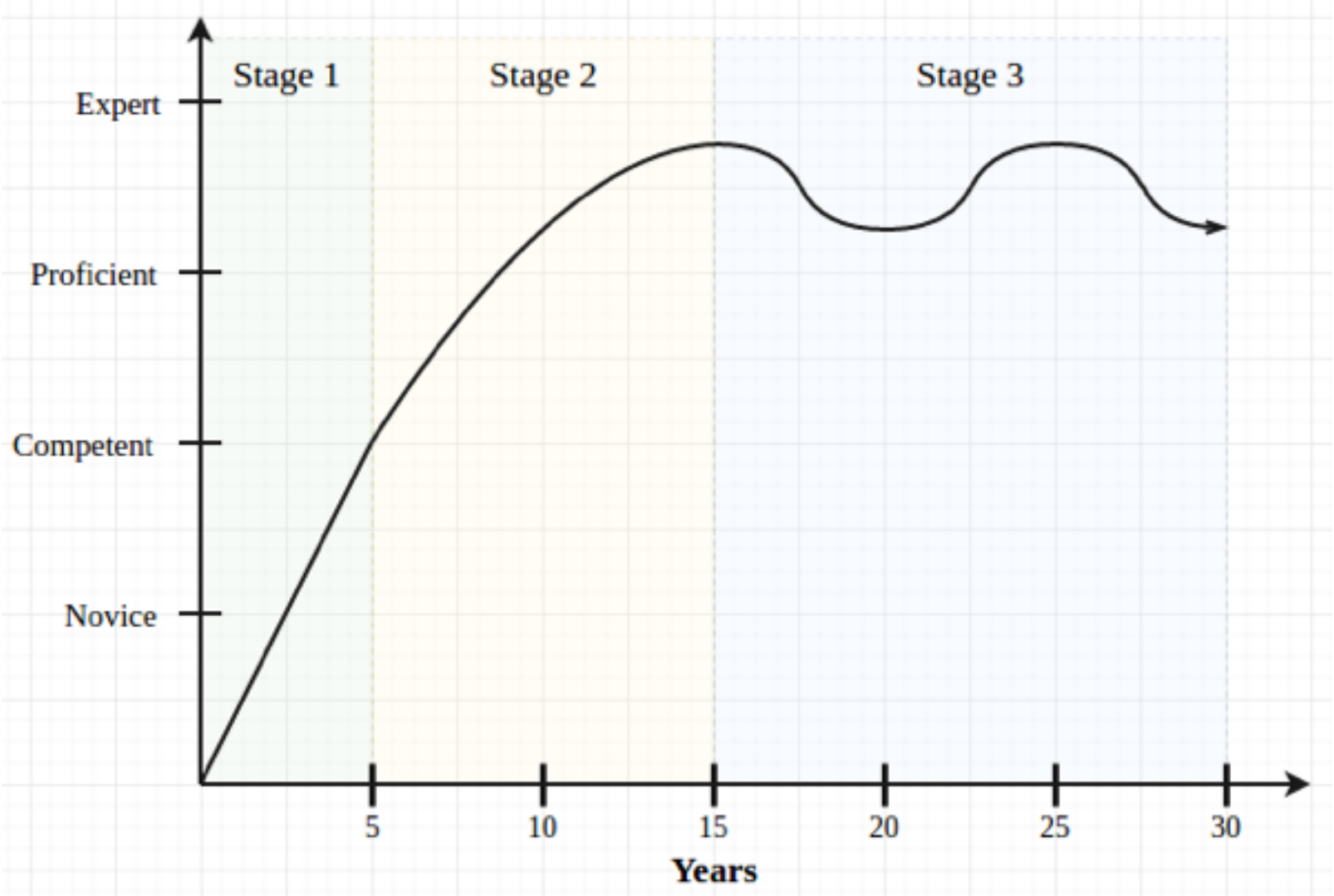
> Half of what a programmer knows will be useless in 10 years.

This could be way off (and there are many caveats of course - read on!)...but it seems about right for me. If I learned nothing else from this point forward, I bet that only about a half of my knowledge could I still use in 2026 (long live SQL!), and the other half would probably be of no use (React Native, perhaps?). Now of course I *will* be gaining new knowledge to replace the dead stuff, but will it be enough? Will I know more (useful) knowledge in 2026 than I do now?

This brings me to the second concept, **knowledge accumulation rate** - the pace at which we add new things to our knowledge corpus. In every field, there is a certain threshold of knowledge that must be met in order to be "certified" (or at least hireable), and the early portion of a career is typically dedicated to acquiring this knowledge. In programming, however, because of the fast decay of knowledge, it seems like we never really transcend the "student" period. We know we must always be learning, and this makes the stages of our career a bit atypical.

The Three Stages

If I were to graph an average programmer's knowledge over the course of their career, keeping in mind knowledge decay and accumulation rate, I think it might look like something this:



In the beginning of our careers, in what we could call the **eager apprentice** stage, accumulating knowledge is relatively easy. Everything is new, and so each experience is a vehicle to gain more knowledge. Moreover, since we're younger, we often have fewer hard obligations, and so we probably don't mind spending a few nights and weekends picking up new languages and frameworks. Lastly, and importantly, the expectations on us from our employers is lower. Everyone understands that we're junior, and so more easily than our colleagues, we can carve out a little time during the work day to fill in holes in our knowledge. This is a fun stage, but there's this persistent feeling that there's so much we don't know.

At some point though we cease to be novices, and we establish ourselves as productive, self-sufficient developers. For the first time, the gap between us and our colleagues (even the ones 10 years our senior!) does not seem so large. This fills us with vim and vigor, and so this is the **rising star** stage. The investment we made in learning has paid off, and just about everything we know is still useful - i.e. none of our knowledge has noticeably decayed. With this reservoir full of relevant knowledge, we begin to earn the respect of clients, peers, and managers, and with this respect comes titles, salary, and opportunities. Though we don't necessarily see it at the time, this is also an important point of inflection.

It's at this point that two things happen. First, that promotion to "senior" comes with something more than just money: greater expectations. Employers need their star programmers to be leaders - to help junior developers, review code, perform interviews, attend more meetings, and in many cases to help maintain the complex legacy software they helped build. All of this is eminently reasonable, but it comes, subtly, at the expense of our knowledge accumulation rate. The time we used to have to read tech blogs: gone. Second, it's also at this point that we first experience (or at least recognize) a little knowledge decay. Some of what we learned early in our career is now out-dated. All that time "we" (read: I) spent learning GWT? Lost! Essentially, both forces, knowledge decay and knowledge accumulation rate, begin to work against us.

It's at this point where we enter the third and final stage, the ebb-and-flow of the **steady**

veteran. We are knowledgeable and productive, yes, but we also understand that we may actually know fewer (useful) things than we did at a prior point in our career. A non-trivial amount of our knowledge has decayed, and we may not have had the time to accumulate enough new knowledge to compensate. This can be frustrating, and I think it's why it's at this point that so many of us bail for other pastures - management, sales, testing, or (my dream) farming. We realize that it'll require real effort to just maintain our level proficiency - and without that effort, we could be *worse* at our jobs in 5 years than we are today. There is no coasting.

Humble Advice

This is where I'm at. I still love to learn, but I appreciate that without some herculean effort, I will probably always remain in an equilibrium state hovering around the lower boundary of "expert". I'm ok with this, because I enjoy my personal life more than I want to be the next Martin Fowler (although I bet Martin has a kick-ass personal life too - that guy is amazing). Thinking about my career in terms of knowledge decay and accumulation though has changed my perspective a little.

First, I try to **take the long view**. I'm more wary of roles with excessively taxing expectations and few opportunities for novel experiences. I've seen quite a few colleagues take the bigger pay check at an employer where there'll be little opportunity to work with new things and learn. In 5 years, they realize that much of their valuable knowledge has evaporated and their pay is way out of whack with their *actual* worth. In some cases, I think making less money in the short term (at a better employer) will yield more money (and stability) over the course of a long career.

Second, given that time is limited, I try to **invest most in knowledge that is durable**. My energy is better spent accumulating knowledge that has a longer half-life - algorithms, application security, performance optimization, and architecture. Carving out niches in these areas, I hope, will better bullet-proof my career than learning the newest, flash-in-the-pan Javascript library.

In the end, perhaps I haven't really forged any new ground here, but it's been useful for me to think about my career in terms of these two things: knowledge decay and knowledge accumulation. I'd love to hear any thoughts you have!

Other Posts

- The 3 Motivational Forces of Developers
- The User Interface and the Halo Effect
- Pair Programming - My Personal Nightmare

I believe that software development is fundamentally about making decisions, and so this is what I write about (mostly). I'm the owner of Highline Solutions and also the Principal Technical Consultant. I have two degrees from Carnegie Mellon University, most recently one in philosophy (thesis here). I live in Pittsburgh, PA with my wife and 3 energetic boys. Subscribe here or write me at ben dot northrop at gmail dot com.

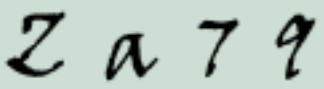
Got a Comment?

Name:

Website:

http://

Are you human:



Comment:

Post

Comments (110)

Tim Daly **October 06, 2016**

Find an iron rice bowl project. Look at open source projects that match your interest and appear to have long term goals. Contribute to that project. This project will keep your programming ability fresh and give you some continuity when you switch jobs. It will also provide something to do when you get really old.

https://en.wikipedia.org/wiki/Iron_rice_bowl

Ben **October 06, 2016**

@Tim - Good point - that's a great way to keep skills fresh. And I had not heard of the term "iron rice bowl" before. Thanks!

Jeff Boes **October 06, 2016**

I'm also a professional programmer, a few months shy of my 60th birthday. (Yeah, that makes me a true dinosaur: I literally learned to program in college on punched cards, and now I work on the web. Along the way I've done so many different things (mini-computers? client-server? SNOBOL?) that I can't be bothered to list all of them in my resume any more.

And I've just been laid off, this time from a job where I'd been for 11 years. This is my sixth layoff (the first was eight months into my very first programming job). This time was pretty painful, because the job I'd been doing for 11 years was the main thrust of my small company's business, and suddenly there wasn't enough work in that line to keep me busy.

So not only do I have about 11 years accumulation of stale knowledge, now I need to leap into the job market and convince someone that my main skill of *adaptability* is worth paying for.

Stefano Borini **October 06, 2016**

@Tim Daly Some people when they get home from work want to do something else than work.

Ben **October 06, 2016**

I just want to qualify (because I'm taking a little heat on reddit :)...I'm being completely tongue in cheek about the "old" thing. Though we all can probably agree that the distribution of programmers skews young...it *is* still a distribution...and so there are plenty who are writing code in their 50s and 60s. Kudos to them.

@Jeff - thanks for sharing that and best of luck.

Victor **October 06, 2016**

There is a knowledge, and there is a phone book. It's possible to know Physics or Math (while there are new things every day, the base stays), but it's not possible to "know" a phone book. Sadly programming nowadays is a phone book which is updated on a regular basis. Yes, there are algorithms and design principles, and they stay, only in the era of off-the-shelf libraries and duct tape used to bring them together they don't matter. Many (most?) algorithms and principles in use today were designed in 60s-70s-80s.

Steve Smith **October 06, 2016**

Nice article. I've tried to focus my knowledge focus on areas of software development that tend to have a longer shelf-life (similar concept to your half-life). To that end, a lot of what I train and present on are not tied to a particular technology version number (e.g. see https://www.pluralsight.com/authors/steve-smith). I think focusing on design patterns, programming principles, etc. in addition to a programming stack with some longevity and market share (e.g. .NET, Java; not just the latest JS framework) can be a valid strategy to combat the knowledge decay you cite here. My $.02.

Raoul **October 06, 2016**

Doctors don't have to worry about their knowledge evaporating?! Seriously?!
You clearly have never been into medicine enough to see it, and I do not blame you for it, but I thought you should know that the medical knowledge half-life is currently about five years. That's right. Five years out of medical school, half of your knowledge is already not considered current anymore.

Matt **October 06, 2016**

Really rings true to me. I am "old" like you. Thanks for writing this as it made my day.

Eric **October 06, 2016**

I am in embedded software. Here the chips that i need to know quite well go obsolete quickly. Each new assembly language is similar enough that the learning curve is short, but the learning continues on at a fast rate.

Stangles **October 06, 2016**

I'm not quite sure I agree with the parallel drawn between programming knowledge decay and knowledge decay in other fields. I would imagine that knowledge decay exists in other fields and may be just the same as or worse than in any software field. Isn't it feasible that we would have a major overhaul in the way we understand biology? Even if that never happens, I'm not

sure all the "front-end fad frameworks" really equate to a "major overhaul" of understanding in the programming world.

I definitely agree with your point to invest most in knowledge that is durable. The fundamentals of programming (computer science fundamentals, essentially) provide a valuable base knowledge that equips you to understand higher level abstractions. This ultimately is what informs your ability to learn and stay relevant.

Ben **October 06, 2016**

Thanks for all the comments. Really interesting points.

@Victor - Thanks! Agreed about the "phone book". Interesting analogy.

@Steve Smith - Yup, I've come to the same conclusion. Thanks.

@Raoul - We'll have to agree to disagree here. :) Though I admit I have absolutely no data to support this (!), anecdotally, I have a handful of friends who are doctors, and we've talked about this. My take is that they have 99 problems but worrying about keeping up with a 25 year old who knows all the new tools ain't one. That's just what I see, but I could definitely be wrong! In any case, maybe I shouldn't have picked medicine as my example. I was just trying to make the point that I think knowledge decay is a thing, and it's arguably more pronounced in our field.

@Matt - Thanks!

@Eric - Interesting. I've never done embedded stuff, so I wasn't sure of the "decay" in that area.

@Stangles - Yeah...I'm not totally sure either about other fields. My perception is that decay is faster in our field...but I could be off. When I talk to friends who are in other fields, they don't seem to feel the same pressure to "keep up" (though they feel other pressures!).

Paul **October 06, 2016**

The perception that "40 is old" is a creature of Silicon Valley and it will change when the chief supporters of it age out themselves. The fact is that frameworks have the lifespan of mayflies and if that's all you know, you're right: you're toast. Every survey of language use for 30 years has had the same 3-4 languages at the top: C, C++, Java, and usually some functional or scripting language lurking at #4. Outside the web development bubble, those languages are not only how work gets done, but how the frameworks themselves are written (someone has to create them, you know). What you have that is durable is the knowledge of HOW to develop code. By the time you have experience with 6-10 languages, you understand they are 90% the same. If you don't have significant experience in 1+ of those top languages, you are a glorified script kiddie and can't deal with what's under the hood of the framework of the week. That kind of knowledge never goes stale.

Just this week, I saw a post that suggested the number of frameworks and components on github is over 170,000! There is no such thing as an expert for that mass of junk. That isn't what makes you an expert developer and anyone that thinks it does is simply ignorant.

Unfortunately, too many decision makers don't know the difference. The fact is that the flavor of the week is never universal at any scale. React Native is the buzz in one corner, something else is hot over there in another corner. Don't define yourself that way.

BTW, I was first paid for coding 40 years ago.

Rajesh P **October 06, 2016**

I am going through the same phase where I feel I am stuck at a good paying job with little technical challenges. I stumbled upon your article just when I was going through this introspection about my career. Thank you for providing the wisdom my friend!!

Elf M. Sternberg **October 06, 2016**

39. 39!?

Pfft. I'm 50. I still work in Javascript and Python every day, and I still contribute to a variety of open source projects. The real secret is simpler: I like what I do. I have a variety of interests that require software to solve them, and if they're small enough I'm going to write that software in a way that teaches me something rather than just take down an off-the-shelf product.

But you know what's really missing from these flash-in-the-pan businesses scattered around San Francisco? Knowledge about things other than the web. Secondary storage, garbage collection, domain specific languages, parsing in general. Knowledge about professionalism. I've worked at startups where teaching these kids about pre-commit hooks and linters was considered a miracle-- go fast and break the build!

Sasha **October 06, 2016**

Ben, thank you for sharing. You spoke right to my core since I'm going through the same thing. Have to stay relevant. That's the name of the game. Good point about SQL))))

Tom Harrison **October 06, 2016**

I am 54. I am a developer, engineer, programmer, architect, bug finder and fixer, QA, ops, and all the rest although 98% of my time is spent being a manager. My wife is a couple years younger and spends all of her time as an engineer designing and building software. We rock.

Fear not -- what we love about software is that it has a short half-life. We must learn constantly to survive and thrive. It doesn't get easier, and the new stuff is coming faster, I think. But life long learning is what makes software so amazing.

But some of what we learn is timeless. Recognizing patterns so we can avoid reinventing wheels (and use. Martin Fowler's and the others' patterns) still apply. Exceptional listening skills still help me avoid pitfalls in design and architecture. Understanding the DNA of networks and protocols and all that gives me superpowers. I can diagnose problems, debug by reading code, and know where to look for logs in seconds, skills that even the 15 year veterans are still grokking. And one thing never changes: we programmers are dense about how "normal" people see the world -- the greatest skill of an engineer is communicating with

ourself and with others.

Whether the 29 year old who is likely to be hiring me for whatever my next job is knows it, that's another question indeed :-)

Peter Job hn **October 06, 2016**

I have been programming for 45 years. Programming is understanding a problem, finding a solution and implementing it. That knowledge never leaves. Each programming language is a few weeks to learn and a few months to master. Most of them are very similar. And the same goes for OSes and DBs. The problem is that many people believe that old dogs can't learn new tricks. Ain't true.

Dennis NG **October 06, 2016**

Move to management but still do some coding (like push a small app to Apple Store kind of). From Apple II, S/390 assembler programming, ... I feel the pain. But to be honest, we are more stable now for a decade after Java comes out. Know Java you are fine in corporate. It is the new COBOL (and work similarly in terms of maintenance, lots and lots of verbose code and that create jobs). In the other area such as when I pick put the objective C decades ago during the Next transition, the framework has more and more feature but the core is still objective C until lately of Swift. Most of the jobs I asked for is related to C family (C, C++, Java, objective C etc.)

Of course in the front end, there is a lot of alphabet soup and framework which deposit to our systems over the year which I always wonder, what on earth can anyone know all these. Are they or am I serious? But they have to write down, as you cannot throw away application that work.

The application side is impossible. Still, I have been on the system side actually closer to the iron. For a decade I were managing 3380 disk, IPL, S/370 assembler language, ... etc. Not vacuum tube but ... Whilst there is a lot of changes on this area, the core has not changed that much (except DEC is gone). You are still using similar technology (which you can grow with) like Cisco router, (not SNA as Mainframe is out by now), Linux/Unix, ... Yes you have hyper-visor and software defined network ... but you can learn. Just for fun, I read the latest Z thing and it sounds very familiar. The Cisco router I look at is also not changed that much. Hence, whilst agree with you, there is still some corners which you can hide. It is changing very fast but it is not that bad in some corner.

But on the front end ... good luck.

Escape to the management is not fun, all just human, cost, time, ... issues. I guess those that can be done by computer and programming has tried to do it there more and more. Left those (even in programming) be managed which is messy and hard to deal with. But one may say that is another corner where the basic has not changed. Presenting powerpoint to your boss may be no different from a speech by a slave like MOSS to the slave head in Egypt. Show your corba I showed mine etc. Human job are hard job.

Steven Almeroth **October 06, 2016**

Great post; I loved every word of it. I think the important thing is to do what you love; if it

brings you money then you're blessed. A family instead just needs time. Well, a little money helps.

Jim Phelps **October 06, 2016**

Nice article, I can relate on several of the points. Speaking as a 40 year old programmer myself, I don't think I'm old and I'm pretty sure my 40 year old peers don't either. Is this article written for people who consider 40 to be old? I remember having that mindset in my 20's and maybe early 30's.

Chris Chiesa **October 06, 2016**

I'm 53 and what I've learned to do is leverage the "next level" of knowledge gained in my not-at-all-misspent youth, namely, *how things actually work, deep down* and *how to make very different things work together*. These are skills I learned in the 80s when there were no IDEs, integrated software suites, out-of-the-box solutions to everything under the Sun, PCs, GUIs (where I was, anyway), etc. If you wanted that stuff, you could have it, but you might very likely have to "roll your own" - - write it from scratch - - and we DID, in many cases. I worked on the user interface and utilities for a Digital Image Recorder, which involved everything from low-level tape and disk I/O, to sussing out foreign tape and disk formats, to implementing a reader for the original Macintosh filesystem on VAX/VMS, to cobbling up utilities to automate inbound-and-outbound dialup modem pool sharing, to... well, you name it. Most of that was in Assembler, with some Fortran, and a little C thrown in the last two years or so (of nine). In my next job I learned most of Unix, most of the X window system, C++, and Perl more-or-less simultaneously in the first few months, then spent nine years creating (often multi-threaded, once even also multi-headed) GUIs to control, and receive data from, one-of-a-kind scientific instruments intricately interconnected using an eclectic mix of both new and thirty-year-old technologies. Then it was off to doing custom video-image manipulation on custom hardware, with its own in-house GUI environment, in a weird colorspace, and becoming intimately familiar with the Linux boot process and USB Wifi (which was new and somewhat flaky, at the time) functionality. Then more custom hardware with yet another in-house GUI system, plus GPS and a whole lot of other stuff. Then off to my first position programming on Windows, updating and developing apps in 2011 using tools that came out in 2002 (in the other room they were programming-and-debugging firmware using RS-232 serial connections and ProComm). Lastly, it was off to a non-software-development job, slinging medical data across TCP/IP, and enhancing tool functionality with my own Perl programs and don't-ask-don't-tell investigation of the details of some of the software involved. I have stories that would curl your hair - - or at least, that of the young apprentice sitting next to you...

The last new programming languages I learned were C++ and Perl, in early 1998. I've studiously avoided every technology and silver-bullet programming paradigm that's come out since then. I know next to nothing about Web or Mobile programming. I've felt kind of bad not to have kept up, but on the other hand I've never been short of work, once I locate a job that DOESN'T require Web or Mobile languages-and-tools...

I've recently been up for a couple of jobs that are an astoundingly perfect fit for my exact combination of experience - - slinging Healtcare data while picking apart old tools and databases and refining/improving them into new ones, and stuff like that. You don't HAVE to know all the new stuff to still be employable. (I may also have been relatively underpaid fo my years of experience; but then, I've been in small markets exclusively, and don't consider myself a rockstar to ask for the big bucks, though other people often seem impressed with what I do.

Robert **October 06, 2016**

I am a 60+ promrammer, still 100% programming for my daily bread.

A few things some of you younkers don't get, I think:

(a) IT is, was, and always will be, a fad-driven industry. Reason don't come into it. Don't expect logic or reason to be the drivers anywhere.

(b) All this talk about "investing" in, say, learning Hot New Framework X. It's all BS. You are not "investing" anything. You are SPENDING. And you wont get it back, so be careful what you spend your time on.

(c) Last point (and I know this is controversial): You are in IT to make software that other people are prepared to pay for (maybe indirectly, but I think you can follow the whole "payment" idea). That's it. You choose how best you can do that, to maximize your own return. If you decide that you do it best by using Hot New Framework X, then by all means, Framework X away.

Akash **October 06, 2016**

Don't know whether I can be called myself as a programmer. 8 months of experience and this article as well as the valuable comments helped me a lot to decide about where should I spend most of my time. Rather than learning more about the framework that I am using now, I am gonna spend more time on algorithms which might come in handy in the future. Great article and great views in the comments section.

Nigel Hamilton **October 06, 2016**

I'm a 44 year old programmer.

One of the best tools a programmer has to help manage their knowledge is on their desk!

The earlier you start writing code to help yourself code the better.

Robert #2 **October 06, 2016**

Please fix whatever is injecting all those \\ characters into comments.

Nigel Hamilton **October 06, 2016**

Another thing ...

Just like in other disciplines deeper expertise and wisdom grows over time.

This isn't the new framework fluff that gwasshoppers gravitate too.

It's the deep knowledge gained after iterating through thousands of problem spaces - knowing what worked, what didn't and understanding why.

Mario **October 06, 2016**

Hi Ben. Very nice and true article. I'm currently 38, working on Information Security, which is probably as much or even more continually changing that the development ecosystem. Sometimes I do feel like extremely old when I talk about things that my colleagues have never ever heard of, despite with my youngest colleagues its roughly 10 years age difference. This situation at some point led me to think that my knowledge decay rate was unacceptable and I should spend more time learning and less time �managing�.
Now with a bit more of introspection of who I am and what I do and how well I do it, I try to take advantage of this experience and the perspective that time provides me to further focus on making decisions instead of concentrating on the immediate. I do continue learning and being proficient on the �latest� but I stay a bit more on the surface than some colleagues which go deep into some topics.
In the end, life is what happens when you get out of your computer..

Christian H�ning **October 06, 2016**

Thanks for this post!
I especially liked the part about investing in lasting knowledge like architecture, algorithms and that kind of stuff. You gotta have the ability to grasp new tech fast nowadays but without proper understanding of basic (and not so basic concepts) this is rather worthless, since you cannot fully leverage the new and shiny stuff.

Also I like to add an analogy: I've worked at the movies as 1st and 2nd AC (assistant camera)for 6 years prior to entering computer science. Thinking about that I found there to be much of a parallel development. The 'concepts' of how movies are made essentially are the same as they were som 20, 30 or 60 years ago (albeit faster, more vivid etc today). But today people working in the filed have to catch up with digital evolution every year. New digital cameras with new menus and stuff, new codecs, new tools, apps continue the list.

I am not going to say this is bad, but from a getting-things-done point of view you could shoot a movie as good on 35mm film as you can nowadays on 4K digital raw flash drives.
Thing is: Without knowing how to light things, where to put the camera and how to visually support the narrative (read: support the customer's need ;-)), it doesn't make a difference whether you do this on 16mm, 35mm, 4K or nifty, fancy, shiny new 16K with a gazillion colors ;-)

Cheers

Ben **October 06, 2016**

Also about to hit forty. Made the same "mistake". Colleagues who stuck with one language the last two decades e.g. C++ are now �ber experts working on mind blowing projects for astronomical money because they have spent twenty years deepening and deepening and deepening their skills and knowledge of that eco system. Worth thinking about.

Mark Koennecke **October 06, 2016**

At 56 I have to be considered old as a programmer too. In my experience all those years and fads leave some basic experience behind. A new programming language, a new thing, you pick it up faster because you have some fundamental experience and know what you have to search for when questions arise. Or where to look for the bug you are trying to fix.

I am also in a happier position that I am not a mere coder but also a domain expert. So, this might be something to strive for.

Kshitij **October 06, 2016**

Thanks for writing this post, which has all the stuff I keep thinking about as I am also one of the older programmer. Best of luck for your learning.

StewartMackenzie **October 07, 2016**

Fractalide is a FOSS project, github.com/fractalide/fractalide and is meant to be an Iron Rice Bowl project. Rust occupies a space that C occupies so it's pretty long lasting. Secondly, the hopes are that an increasing amount of components that specialize refine and are regularly reused means knowledge becomes solid and is built on previous knowledge. Hence this system is designed for long life. Jump in and participate.

Peter **October 07, 2016**

I started programming in my 30's, I am now in my 50's and so have about 20 years of experience in software. I spent previous years in electronics and a couple of years working as a metallurgist. So what I want to say is, think of it as starting your career now and when you are in your 50's you will have "another" decade of experience

I personally found functional programming knowledge to be "decaying" quite less fast. In fact you build up on it. **October 07, 2016**
All this framework bullshit you mention above can be avoided. There are solutions, just look around.

Tsega **October 07, 2016**

I can't believe how you nailed it for me! I couldn't have put my experience in better words that what you have done. For years now, I've been telling my wife that when I take on projects almost half the time is spent reading up on the latest technology/trend on how to complete the project. She's a dermatologist, she hardly needs to refer back to her textbooks or scour the internet to learn the latest technique to clear acne dark spots. She already knows!

I've kept the "student" hat on for a while now and the "JavaScript Fatigue" is getting to me these days; I want to be an expert but before I get half-way through the API, the damn library is obsolete!

Thanks for sharing your insight, I feel I'm not alone here and your resolve to start focusing on niches that have longer half-life is invaluable!

William Payne **October 07, 2016**

I think that there is some room for strategic positioning here.

SQL isn't going away. Unix and Bash aren't going away. Python and C aren't going away.

Yeah, the world of front-end web-development is crazy volatile -- but that's not the only world that there is. Embedded is huge. Data engineering is huge. Back-end development is huge. All of these areas of expertise are built on much more stable foundations -- foundations that have lasted decades and that stand a (p>=0.75) chance of lasting for decades to come.

Plus, there will always be people who need problems solved and don't give a damn what tool is used to solve them.

Finally, consider this: Fashion moves in circles. The next hip thing is always a rejection of the last hip thing. The details may be unpredictable, but humans do not change in their fundamentals. They are always fighting a battle (with religious fervour) against whatever picayune issue last grabbed their attention, and the experienced professional will always have been around long enough to see both sides of the fight and to move the soul of the organisation (by inches) towards a working consensus.

This is the value-proposition that you (we) need to get good at selling.

Nikhil Vibhav **October 07, 2016**

Great read Ben!

I'm still at the novice stage that you mentioned about and I must agree that all knowledge has half-life and accumulating knowledge with larger half-lives are the way to go forward. :)

Munawwar **October 07, 2016**

Good post. Interesting point about "knowledge decay". Today I don't use many things I learnt as a student (C, Java, PHP, GTK... Win32, Flash, VB are dead). Yet many concepts/ideas I saw has stayed.

I wish more people would give the advice to everyone to learn about underlying systems/concepts used in the "latest hot thing". People really need to think about the longevity of their knowledge. And also how to contextualize that knowledge for a new problem, and the pros and cons of these systems.

"Algorithms, application security, performance optimization, and architecture"
Yea..there are lot more though. Specific domains like networks/TCP, databases, operating system tools & features, infrastructure security, hardware. Also some alternative fields like machine learning, computer vision...heck even marketing and people management (which some may have forgotten, that every programmer is affected by the business as a whole). There doesn't seem to be an end. I think that's how to stay relevant.

James Phillips **October 07, 2016**

My theory is that knowing only *how* to program without knowing *what* to program leads to the ongoing proliferation of "frameworks du jour". For people with programming ability only,

what else are they going to program?

Ben **October 07, 2016**

Wow! Thanks for all the good thoughts.

@Paul - Great points. Agreed that there is an "immutable core" of knowledge that is more important/valuable than the latest/greatest framework. However, I need more than that core knowledge to get my client a new hybrid mobile app up and running.

@Rajesh - Thanks!

@Elf M. Sternberg - Haha. I know, I was being completely tongue in cheek about 39 being old. Kudos to you for still finding enjoyment in programming at 50. All good points.

@Sasha - Thanks!

@Tom Harrison - Great comment. Agreed that there are both hard skills and soft skills I've picked up over the course of my career that are invaluable, and that I'll never lose. And I like the perspective you shared about embracing/loving the fact that the industry changes. Agreed that this is what makes programming so interesting - the novelty.

@Peter Job hn - Thanks. Props for 45 years of coding!

@Dennis NG - Cool story. Thanks.

@Steven Almeroth - Thanks - much appreciated. Agreed - getting paid to do something you enjoy is awesome.

@Jim Phelps - I was just being tongue in cheek about the "old" thing - though where I work, being 40 probably does put me in the top age quartile.

@Chris Chiesa - Great story. Thanks. Agreed that you don't need to know all the new stuff to be employable...though I do think it helps to stay up to date a bit in case you find yourself on the job market.

@Robert - Good points. Per your (a), I *totally* get it. http://www.bennorthrop.com/Essays/2015/slaves-to-fashion.php To your (b), I have to disagree. When your customer wants a fancy single-page web app, knowing the latest Javascript framework is essential, I'd say. For (c), good point...but it's often the case that someone else has chosen the framework, and to stay productive you need to dig in and learn. In any case, great thoughts!

@Akash - Glad you find value in it.

@Robert #2 - I will. :)

@Nigel Hamilton - Thanks!

@Mario - Thanks for sharing that...especially the last part. :) Agreed about the less time "managing" and more time learning point.

@Christian Huning - Interesting parallel to movies!

@Ben - Very interesting point. I do sometimes I wish I focused on one area. I was information systems rather than computer science...and I think the world of application development is just inherently more chaotic.

@Mark Koennecke - Good point about the domain knowledge.

@Kshitj, @StewartMackenzie, @Peter - Thanks!

@Tsega - Thanks! I'm 100% with you.

@William Payne - Great thoughts.

@Nikhil - Thanks!

@Munawwar - Good points. Agreed about data science. This is a great specialization that seems to be a little more durable.

@James Phillips - Good point.

Adam Tuttle **October 07, 2016**
I wrote up my thoughts in a blog post: http://adamtuttle.codes/the-cost-of-abstraction/

J.T. Grimes **October 07, 2016**
One thing I've discovered is that as we get older, learning becomes more difficult. That's just how brains work.

Part of why a lot of people "age out" of development is that accumulating knowledge at the same pace we did when younger is an order of magnitude harder and it stops being worth it.

Jean-Philippe Boily **October 07, 2016**
That was a really refreshing post! I just reached a point in my career where it just seems I don't know anything anymore. I looked at job offers and read tech blogs and there so many diverse technologies used by everybody that it's enough to get you dizzy. That's why the more I think about it, the more I tend to give more importance to the management services in my consulting services.

Thank you for this it feels great to know that I am not alone to feel that way :)

Dave **October 07, 2016**
62 and still learning new frameworks, but that is still easier and more lucrative than almost any alternative.
Since you mention farming, watch out for the mid-life crisis that seems to hit in your 40s where you abandon everything and switch careers and maybe partner etc. Very destructive - get yourself a hobby farm instead and keep on programming.

Brian **October 07, 2016**

You mention Doctor's not having to worry about knowledge evaporating. That's not entirely true. If there are practices based on biased or false studies, you bet your ass there are practices in use today that should not be in the future.

One area Doctor's usually have limited education in is nutrition. There have been many biased studies on nutrition that caused several generations to think they should eat one way, when it's backwards. Doctor's today still preach incorrect information about nutrition. And it'll be a long time before we undo those mistakes.

Does that type of blunder occur in the software industry? Possibly, but to different effect.

M. H. Neifer **October 08, 2016**

Adrian Kosmaczewski has an interesting article on Medium about old programmers. https://medium.freecodecamp.com/being-a-developer-after-40-3c5dd112210c#.znjwmnr7b

Glenn Larsson **October 08, 2016**

I tend to agree, but the total image of all the knowledge gained makes me a better programmer, even what will become "crap" in a few years, is valuable experience in thinking.

Sometimes i've identified a problem with a programming language and i've written stuff to compensate for it. Without that knowledge on how things can be made better, i would suck and would write crappier code.

While it is true that i have no use for my DBase and MS Access knowledge, when moving on, i had a good understanding on how databases work, and when i learned MSSQL and MySQL the learning curve wasn't so steep. Same goes for learning Teradata (by myself) and even NoSQL like Cypher - the teacher used SQL as a reference to explain what things did.

If you say to someone that you code in Visual Basic, you are generally seen as a crap programmer even though you know multiple languages. And while i rarely have met any VB coders with any mentionable skills, it's not their fault. I coded in VB 3.0 to VB 6.0 and while C++ had MFCs, i had to write includes for DLL files and write wrappers for everything but the kitchen sink. Now with .Net (which i picked up by myself), most of that is gone, but i sometimes still have to do that when .NET fail to provide an interface to a API function in the Kernel.

As a Forensics analyst you need to learn new things all the time, and that field is more product centric. If a product dies, you risk becoming obsolete - unless you know how to write code. As a former Network guy i had a lot of use of my programming skills, i wrote scripts and tools to help me do my work more efficiently, and today a lot of networking guys learn powershell to do their job (which is basically "Commandprompt".NET which many of those guys are unaware of)

TL;DR: What i'm saying is that by knowing one thing, you can use that skill as an intermediate to learn something else, or even do something else more efficiently.

42 year programmer (and other things), started coding @ age 11.

Nah **October 08, 2016**
What a fantastic read. Thank you.

The lower on the stack you go (cpu, kernel, os) the longer the half life. The higher you go (app, front end) the shorter the half life. Fundamentals will serve you well.

I suck at programming **October 08, 2016**
You were lucky dude. these whizzkids wizzing past you now will be jobless in 30 years when AI and algorithms take over their jobs.

HoboProgrammer **October 08, 2016**
I'm in the same boat, what kills is me when the recruiters ask me how many years of angularJS experience I have, on my resume if mention I have 18 years of javascript, duhh. I'm seeing lot of startup pick a new technology and ride with it, which is great because it's innovation at work, but what I don't get is when these new companies want 10 yrs of angularJS experience ???

M. H. Neifer **October 09, 2016**
Interesting news on this topic: More software engineers over age 40 may join a lawsuit against Google
https://www.yahoo.com/amphtml/finance/news/more-software-engineers-over-age-195308179.html

I suck at programming **October 09, 2016**
so front-end is dead already
http://www.techradar.com/news/internet/wix-s-ai-is-the-web-designer-we-didn-t-know-we-needed-1322717?

Jim Davis **October 09, 2016**
I got in to the IT(DP,EAM,EDP,MIS,etc.) in 1954 where I worked until retirement 41 years later. I progressed thru every position in the field, the majority as a programmer and later management. I've seen the field grow virtually from its birth until now. That being said I can relate to your article and feel sad at times that so much of my knowledge has become obsolete. Those early days were really great!

HS Coetzee **October 09, 2016**
I have to say, this was spot on.

Don't really have anything to add, except that misery loves company :)

David **October 09, 2016**

Hi Ben, thank you for posting this. As someone who may be starting their career in programming and web development, it's nice to hear a perspective from someone well into in,

Harold **October 09, 2016**

I ya, been there done that 7-13 times now.
..
My current project is a "SLOW-SUPERCOMPUTER" instruction WORD of
4096 Bytes. Make your MODELS work
harder, and become SMARTER !
..
Harold P Boushell

Auribau **October 10, 2016**

Very interesting. My dime: There are solid bases to retain for the programming activity: algorithms, basic mathematics for the majority, very advanced mathematics for some topics, patterns recognition. Most of all quoting Dahl or Nygaard "Programming means understanding". The problem is the fashion victim approach that is so frequent in our business. I am 63, I am programming (in addition of managing projects and doing some business) with a language and environment that I have just discovered 11 weeks ago. I replaced a young guy rather brilliant but absolutely not involved with quality, planning and sustaining issues. I think I managed to do this because my fundamental knowledge in software development is solid, that I am not reluctant to have to learn new things and mainly because I am cheaper that the younger guy ...

Kirill **October 10, 2016**

What programming are you talking about? From my personal experience, while the web field is indeed ridiculously unstable (there's always that JS framework of the week with the next week bringing somethng entirely different), system programming stays mostly the same. Sure, there's still something to learn (unless you program in C that keeps ignoring all progress in CS since the 60s), it isn't nearly as ridiculous as web development.

Nigel Pearce **October 10, 2016**

What a great article and completely sums up my view on the profession I chose way back in 1981.

I started writing software because it offered that subtle mix of my two main interests at school: mathematics and art. Designing user interfaces and making them work well is what has driven my continued interest in computing.

And today at the age of 55 I can tell you that I hate having to learn new programming paradigms and languages when the end goal remains pretty much the same as it was way back in 81. I've realised over the years that I don't love programming, I love the results of what I can create with it. For me the tools are simply a means to the same end: good looking well working applications.

So if you prefer creativity over the nuts and bolts of programming I can only offer this shred of advice: try to stick with programming languages that might have a reasonable shelf life. I

managed 7 years with Turbo Pascal, 14 years with Visual Basic and I'm now coming up to 10 years with C#. It is still possible to produce good stuff without resorting to the next great tool, this is my latest C# creation: www.rightbooth.com

Perhaps I've been lucky with my choices and perhaps the rate of change on dev tools was slower last century, but if I knew then what I know now about the volatile world of computing I would probably not have chosen it as a career.

Caleb **October 10, 2016**

I'm 40 and about to move into programming starting with a code bootcamp. It's a huge investment for me and my family, and by investment, I mean "SPENDING" (according to Robert). I'm coming from a musical career and have some design experience so front-end seemed the most appealing and fit with my skills so far. I'm doing pre-work for the upcoming blitzkrieg of bootcamp and I'm enjoying it so far but...

If I'm starting out "old" should I even be doing this? I have been excited and encouraged by a group of musician friends who have moved sideways from the touring musician life into coding and have not looked back. Admittedly, they are all mostly younger, a few the same age.

I had hoped that my creativity, management experience, people skills, and work ethic would go a ways towards making me more appealing in this industry as I come out of the bootcamp gate. Pipe dreams? Am I just going to be seen as a freshly painted dinosaur??

I'm planning to take Paul's advice and learn some back-end which is becoming increasingly interesting to me anyway the more I learn on the front. Is there anything else I can do to "bulletproof" my career?

Amy **October 10, 2016**

At 50 I finally took a permanent job with a super stable huge company making a modest $120k + great benefits. I know at the most I have 15 years left before the pace overwhelms me. I just want to make some great software that'll live on after I'm gone and to bank as much as possible in the next 15 years and then coast into retirement on my own farm. It's been a wild and wonderful ride but you nailed this head-on my friend! Great career path and a wonder 25 years so far, but not for the faint-of-heart!

Tomas **October 10, 2016**

All those "fancy" new JS libraries popping-up these days are not offering anything really revolutionary like they claim. And some of them with their package dependency hell, makes it even worse by inventing some ugly Frankensteins made of a bunch of small libraries created by people with different mindsets. There is even an article written as a joke which makes fun of JS web app ecosystem (hackernoon.com/how-it-feels-to-learn-javascript-in-2016-d3a717dd577f).

Anyway, those JS libraries are usually easy to learn and easy to forget. They are what Uncle Bob describes in his blog post "The Churn" (blog.cleancoder.com/uncle-bob/2016/07/27/TheChurn.html).

I agree that the real value lies in what you describe as "durable knowledge" and what Uncle

Bob describes in his article "The Lurn" (blog.cleancoder.com/uncle-bob/2016/09/01/TheLurn.html)

KBZX5K **October 10, 2016**

If you try to invest time in knowledge that is durable, then what were you doing at a React Native conference?

I'd recon spending time with the wife and kids is more valuable than anything a React Native conference could teach you.

Paul K **October 10, 2016**

Debugging is the #1 skill, and it seems to be ignored in a lot of training.

JimS-Indy **October 10, 2016**

The analogy to a physician is telling. What doctors learn in MEDICAL SCHOOL includes biology, chemistry, anatomy, etc. While those subjects don't really teach a doctor how to treat a patient, they ground the doctor in a basic understanding of how the body is structured and what makes it tick. The rest is experience.

The problem with today's coders is a lack of that basic foundational learning. Thus, each new technology is completely new instead of being a new way to "treat the patient", with the underlying goal the same.

New technologies aren't new. They might be convenient, fast, or sexy, but they're all the same on the inside. If you understand the inside, you can make better decisions.

Steve Naidamast **October 10, 2016**

I completely agree with the author's contentions in this post. As a senior software engineer of 66 I have written and published numerous articles on the same subject for a variety of sites.

The technologies we "need"to work with completely matured in and around 2010 for the Microsoft Community. Interestingly enough, the Java Community is relatively stable in this domain with much less hype surrounding it. So this is more a Microsoft phenomenon promoted by a company that has historically been just as much about marketing as it has been about technology.

What we have today are diversions from these mature technologies that make little or no sense in terms of any benefit to a project and especially the business organizations that they serve.

How many times can you relearn JavaScript to do the same things but in a different manner? What else is needed in HTML markup that suddenly makes the current version unable to present a web Page? How many times do developers have to jump through hoops to maintain any sense of credibility in a field that hasn't really innovated anything in a number of years?

The list of questions could go on.

True enough there have been several innovations such as "ServiceStack" to replace WCF's

more onerous environment but how many actually use these tools? And how hard is WCF really?

The problem is not with the vendors but our own profession that insists that if it isn't using the latest technologies and tools, applications can no longer be developed. However, it has always been this rabid hype in our profession that has consistently prevented it from maturing into a stable and productive field of endeavor. It is no wonder that many capable people want little to do with it anymore. I surely wouldn't recommend it to a young person today.

If at 40 you are staring to feel tired of this merry-go-round wait until you are 50 or older. Slowly but surely you begin to realize the stupidity of it all after coming to the conclusion that we no longer build quality applications but simply regurgitate ever changing technologies to keep of with some mirage of productivity.

Robert Oschler **October 10, 2016**

There's also a huge, terrible trap facing most programmers, especially as they rise in value and talent. It is a very common experience to be continually 'worked to death' by an employer; a never ending sequence of emergencies. Worse, you can end up in this state for a years. During this time, your ability to accumulate new knowledge can drop to zero. At that point you are merely marking time until the day comes when the majority of your knowledge has become obsolete. If that event coincides with an IT outsourcing event or a new wave of cheaper, younger developers, you are in real trouble.

John Harding **October 10, 2016**

OK so I was going to zing you about being older than you... But now I see that was all tongue in cheek...

I agree with many of your points - but a couple of different observations:

a) many of us (I suspect you included) enjoy the field precisely because it's ever changing and there's no coasting. I like to say "you're either getting better or your getting worse - there's no steady state" (yeah, that's also why I don't do catchy soundbites!)

b) I agree with the general shape of your graph but I'm not sure about either the timescale or the size of the peaks and troughs. Taking the latter I'd say it's probably true if one were to gauge "expertise" on knowing every quirk in a language. But it's probably a lot flatter if one were to use a gauge of "getting successful, sustainable software completed". As for the time scale I think with a good background in the fundamentals (preferably from a software engineering or computer science degree) that the ramp up time should be quicker than 10 years.

c) I think the main point I want to impart is that it's that understanding of fundamentals that is most important. The technical fundamentals let you ramp up in new tech quicker. The experiential fundamentals that you gather about team dynamics, following principles rather than rules, a healthy dose of cynicism about buzz balanced with an innate curiosity to understand the trends and to gather the good pieces for use in your own personal approach.

I don't know if any of the above makes sense. But at the end of the day I want to be a part of a team that produces successful work product that is beneficial to whatever goal we're working

towards. No one (aside from consultants) should ever get rich from being an "expert" javascript programmer or an "expert" scrum master - they should be rewarded by the practical applications of whatever parts of the language or process fits for the problems they're solving...

Just my 2 cents worth. I enjoyed reading the post and some of the comments. Thanks.

Alan **October 10, 2016**

I've been doing this for nearly 50 years. What changes and decays is the endless stream of tool details and trivia. The ability to solve problems does not, and that's the value we bring to our companies. Someone will always be promoting another "standard" or "better" tool. Don't "invest" in the details. Spend time developing solutions and paying attention of how it's done by others -- past and present. Let reference sites and others keep track of the trivia.

Gary Boswell **October 10, 2016**

I am a very old , 79, programmer. I started out in 1958 on a IBM 704 using SHARE assembly language. I also programmed an IBM 650 with a 2000 word drum as all of the memory.

The problem is that every time the speed of the computer being used increases by a factor of 4 most of the learned techniques must be revised. Also, as the ratios between the speed of primary and secondary memory change,best practices also change. Now that computer as much faster and memories much larger these problems are somewhat less sever. Also the current programming languages hide some of these problems. But now we have the problem of a programming language of the month. The switch to on-line apps and the cloud have recently introduced many new problems. As another has said, it is like the laws of physics are arbitrarily changed every 10 years. Remember, Einstein only extended the domain for physics, he did not throw out all of the physics that applies to low speeds. Writing for the cloud, vs a mainframe, is more like having to do physics in a new universe that is totally independent from our universe. I do believe that the pace is slowing down somewhat. In the future, our industry will finally reach maturity, and the generation of programmers at that time will look back on these days with envy like we look back on the cowboys of the old west. They had all the fun! By the way I am now retired and programming for the fun of it using Mathematica.

I had a wild ride but I would't change it for anything.

Gary

Dennis Drew **October 10, 2016**

I am 73 and still working full time. Hope I win! Started on a IBM 1620 at Cal State Long Beach in the 60's. Now program in C, C++, C# and Java. Hope to see you when you get to 73! Agree with the article and most of the comments.Retired twice but keep coming back as doing nothing is too boring!

Robert Hodgins **October 10, 2016**

Old? at 40?
I'm 74, earned my living programming
mainframes for 33 years, retired 20 years ago and I still program PC's
for fun!

Terry Slack **October 10, 2016**

I hear you. I'm 48 and I spend a good deal of time reading articles on Javascript development at the expense of my C# skillset. Learned Angular and see the limitations. Now on to React and the tooling is frustrating. Feels like Java on the front end. But React-Native beckons. My dream is a bed and breakfast on a tropical Island and we are going to make it happen. Now if only I can make it through another fast paced environment. Which leads to why is software development expected to be lightening fast compared to other fields? But that is another blog article I suppose.

David Lormor **October 10, 2016**

Ramble warning:

I found this line particularly insightful:

"My energy is better spent accumulating knowledge that has a longer half-life - algorithms, application security, performance optimization, and architecture."

While I'd personally consider myself in the transition from "rising star" to "steady veteran", I've noticed this quote seems to ring very true. Things like React or koa or Elixir or whatever are easier to pick up when you have a solid "CS principles" baseline to rely on (and I haven't even studied CS formally). I've run into so many people that think that the next big "thing" is going to solve all their problems, when, in reality, they just have a new delivery mechanism for their poorly-factored code. I often encourage my peers and "understudies" to spend time learning about "real" OOP, functional principles, design patterns, etc. vs. adding another framework to their "repertoire".

I guess what I'm trying to say is that forming this solid foundation allows me to just hook into the entrypoints of new thing "x" and then write solid code, which leads me to take a different approach to learning new tools - learning the API vs. looking for info on "how to do x"...this in turn feels like it actually makes me more proficient in said tool because I've focused on how it works (and whether it will fit a particular need) vs. rolling with the hype and then forcing myself to get things done with tool "x".

Matt McGuire **October 10, 2016**

I just hit 40 in march and have been working in industrial control for around 20 years now. although the processes stay steady, I worry that the language or frameworks I've been depending on will change course and I will have to produce something my customers find "too different". trying to second guess where the future is going in software development and how it will relate to my industry has left me almost paralyzed to work in something new. which language and framework do I choose that will help my career?
one of the other commenters talked about the embedded market and learning new chips. it is true that chips change all the time, but the language (usually c or c++) stays fairly constant, and so do most of the concepts. embedded is likely one of the most stable portions of the software industry, where (I assume) any sort of web development is the most upsetting branch. niche branches of development are usually long term and with one employer over the course of your career, it does lend it's self to a little "stale" knowledge making it hard to jump to a different employer, but then you also don't have to play the race of keeping ahead either.

Ron **October 10, 2016**

I am a little shy of my 70th birthday, Learned Fortran using punch cards back in the 60s. Built my first computer in the late 70s. Did a lot of machine coding in the 70s, 80s and early 90s. Have been chief programmer for a large manufacturer for the last 20 years.
Agree with Victor that basically not much has changed. There are a few new algorithms but mostly a lot of new languages and syntax that pretty much do the same old thing.

Ronald Anthonisamy **October 10, 2016**

Very well written, and it looks as if you have spoken my mind, only, much clearer, because, i had the same worry and in the (little above) your age group, but now that you have shed some light in focusing on "application security, performance & .., architecture, " than learning the new JS library, I am more comfortable looking ahead.. Thanks a lot.

Marty Landa **October 10, 2016**

If you're "old" I must be ancient! I was coding when you were about five years old. I've always considered myself to be a problem solver. The particular tools used to solve problems, as a programmer, are far less important. We've both been through these learning cycles many times throughout our careers. You're correct, It's just part of the business that we're in.

The experience we've gained over the years is invaluable as it guides us in making better decisions regardless of the particular tools we happen to be using at the time. Perhaps you're not giving experience enough credit as is quite often the case these days.

As seasoned programmers we know we can learn anything we need to learn in order to get the job done. We've done it many times before so that's not the real obstacle. I submit that age discrimination is. So many people in this and other disciplines have a preconceived notion that old is outdated. Now that I'm "ancient" I run into this quite often. Unfortunately it's not something you can fight by learning the latest programming language or framework.

As you get older you'll run into this more and more. My advice to you would be to not give up on that farming dream of yours, "old" man!

dave **October 10, 2016**

Sorry, 40 isn't old. you are just a kid.

63 year old programmer for 40 years

Steve Sadler **October 10, 2016**

Become an embedded engineer. I'm 63 and the only language I've used in the last 20 years for a project is C/C++. About the only 'new' info I have to worry about is how to program the particular peripherals built into the processor I'm using (TI DSPs and ARM family).

Bob **October 10, 2016**

Hello Ben,

Before reaching for the anti-depression medicines and signing up for group therapy, think

about all the other skills you have learned. I’m positive in the last 20 years you’ve learned more than just technical skills.

If I may list of a few “non-technical” skills
* How to interact with members of the opposite sex
* How to communicate with actual words, sentences, and [gasp!] feelings
* Autodidact person who can self-overcome obstacles
* Know when you’re getting into trouble, long before getting into trouble

I’m sure there are others. That you affirm life is more than programming, engineering, coding languages, and frameworks is a signal that you are awake.

We’re no different than the engineer who learned analog record technology or wire recording; only to be replaced by recording tape. What about a radio engineer? 80 years ago a radio engineering job was the ticket! How many of us have used tube tech? Wouldn’t we all love to talk to an engineer who used tube tech?!

Who knows what will be 100 years from now? My guess is COBOL and cockroaches.

Be happy with your family and friends.

Ben **October 10, 2016**

Holy crap! I thought the traffic for this post petered out on Friday, but I came back this afternoon to see 40 new comments.

First, thanks so much for the feedback. I love that it sparked such an interesting conversation, and it's been fun reading all the insightful comments.

Second, I can assure everyone, I am not depressed and am not going through a mid-life crisis. Lol. Life is good, and I feel very grateful for my career, my family, and my friends.

I will try to reply to everyone's comments tonight...and also fix that dang apostrophe issue in the comments...but for now, gotta get back to work!

Thanks everyone!

Tom **October 10, 2016**

I'm a bit your senior and have some reflections to add.

- I remember browsing the internet back in 1992. Yeah, the whole thing.
- I remember the day I couldn't find a new interesting tomb in the computer book store.
- I once wrote a tables based fractal algorithm for my 3D graphics engine because I couldn't justify the numerical co-processor on the 386'DX' :p

Now:

- I think at last count my resume has about 300 acronyms on it.

- The number of new technologies and paradigms has become geometric. There is no way to know it all. Specialise and pray you choice well.
- and don't take this the wrong way youngin's but when I'm hiring university graduates that cannot explain recursion, i think it's safe to say that we have a flood of 'thin' programmers, not deep ones.

Honestly, I've been slowly extracting myself from the industry. You're absolutely right that the educational requirements are draconian. No other industry's cogs suffers the pain we do in this regard. The overhead this incurs is ridiculous.

Further, while what we , as an industry have accomplished is amazing, the quality standard has been lowering with each passing year.

Perhaps this sums things up nicely now:

https://xkcd.com/927/

You'll figure it out!

Cheers!

Nigel Pearce **October 10, 2016**

What a great article and completely sums up my view on the profession I chose way back in 1981.

I started writing software because it offered that subtle mix of my two main interests at school: mathematics and art. Designing user interfaces and making them work well is what has driven my continued interest in computing.

And today at the age of 55 I can tell you that I hate having to learn new programming paradigms and languages when the end goal remains pretty much the same as it was way back in 81. I've realised over the years that I don't love programming, I love the results of what I can create with it. For me the tools are simply a means to the same end: good looking well working applications.

So if you prefer creativity over the nuts and bolts of programming I can only offer this shred of advice: try to stick with programming languages that might have a reasonable shelf life. I managed 7 years with Turbo Pascal, 14 years with Visual Basic and I'm now coming up to 10 years with C#. It is still possible to produce good stuff without resorting to the next great tool, this is my latest C# creation: www.rightbooth.com

Perhaps I've been lucky with my choices and perhaps the rate of change on dev tools was slower last century, but if I knew then what I know now about the volatile world of computing I would probably not have chosen it as a career.

Charlie **October 10, 2016**

I think most of what you say is true. It's not a field for someone who wants to just learn the skills initially and then coast. But then who would want a life like that?

I have about twice the time in as you. At 74, my programming work is now limited to open source contributions, but the learning continues as it must.

I think most people eventually tire of learning the latest new technology. Many of them leave the field because they assume that they have to "keep up" in order to remain. But often, it's possible to draw a line and simply work on the technology that is useful for whatever jobs you select. To do that, you have to get to a place where you can choose your own jobs. By spending the first 15 to 20 years of a career getting to that point, the remainder can be spent learning different kinds of things than just technology.

J Williams **October 10, 2016**

Wait till you've been programming for 35 years... :-)

Good programming is not really about knowledge (knowledge saves time, but hey, we've got google now), I think it's more about skills. The programming languages change, but the underlying skills are what make us good at our jobs, and they generally translate easily to new systems.

Although programming is a frequently disrupted field, I think this problem is becoming rife across all fields (ask the mechanic what he'll be doing in 5 years when all the cars are electric, the taxi driver looking at self-driving cars, the surgeon who has just got the hang of keyhole surgery who now has to use the telepresence robot, the airline pilot who has to take exams every few months to keep his/her job...)

The important thing to remember as you get older is that experience really does count, and us old timers can still hold our own against the hotshot kids who are about to invent the next biggest thing (that we invented 30 years ago... you know the cloud, but do you remember thin client? You remember thin client but do you remember terminals?)

jspark311 **October 10, 2016**

I liked your post, and I liked reading the comment thread. Thank you to those who gave personal perspectives.

I'm in my mid-30's, and (like all craftsmen), I owe my skill (in part) to the patience of older engineers who were completing the knowledge-cycle.

Most of the knowledge we carry was a gift to begin with. To not pass it down in good (better!) condition would be a disservice, IMO.

I know that one day, I will carry so much knowledge that it will be more valuable for me to teach, rather than write code or design hardware. My aim is to anticipate that day with the knowledge that I will die, and my job will ultimately become the nourishment of the next wave of craftsmen. This will be a job to be proud of.

Kishore Kumar G Pillai **October 10, 2016**

Hi Ben, Being a 40Yr old and currently looping the 3 time in career from programmer <--> leader and having a doctor wife , I can only express that the article was found so true to my

career journey as well. I was lucky so far in managing meaningful changes (technology or subject matter) with the descending support of mentors beyond my passion to the work. Thanks a lot for this article.

Chris **October 11, 2016**
I am a 45 year old programmer, had the same thoughts some years ago. I became a teacher and now teach programming. What I have noticed from a decade of both careers is that students are rarely born with **nowse**, common sense, problem solving approaches. They follow an example, it fails because they have **=** instead of **+** and that**s it, it is unfixable without teacher assistance.

I would say to you that you have picked up a lot of **nowse** or problem solving abilities along your journey. You have a lot of underlying skills which makes you an experienced coder and a good potential teacher. An **if** statement is an **if** statement, right? Unless you are a student having listened to the teacher talk about if statements and then types **1F** because the **i** looks more like a **1** in the default Microsoft font and then declares that the example does not work.

You know more than you think. Thank you for your article, from an even older coder!

Bruce Roeser **October 11, 2016**
Old? At 40? LOL! I started my career as a programmer in 1976 and am still going at 58! Young whippersnapper! ;-)

Daryl Lucas **October 11, 2016**
I'm 52, and I could have written this. Well said. Thank you for saying it so eloquently.

Dan **October 11, 2016**
I was intrigued by the title and the beard. As a 62 year old programmer, I couldn't agree more about the ephemeral vs. accumulation.

The ebb and flow is what keeps me going.

Sean Josiah **October 11, 2016**
Ben,

Good Article/Post. Your descriptions and generalities ring true. Though, do not be discouraged.

I would add a reminder and tip for the younger programmers: There are additional skills gained simultaneously on the knowledge trajectory. These impact the rate of the trajectory and affects how the knowledge distills. If you came from or simultaneously worked in networking, systems integration or quality assurance for instance your expertise will be augmented, even if the nature of your programming work is outside those realms. Your overall experience should inform and enhance your implementations and invariably it will. What you learned in other fields can apply if you abstract it. Every job is a lesson. (Construction, Manufacturing, Plant

Operations, Security... whatever)

The rewards are accomplishing the requirements, exceeding expectations, getting paid for the execution of the solutions, mentoring for appreciative peers and forming esteemed relationships. On the way you end up marketing, even if only yourself, inspiring, selling, wowing, documenting and surprising. These never get old. Estimating, predicting, analyzing, negotiating, convincing, correcting, challenging, cost/value etc. Yes you are a programmer but your business value is providing structure and manifesting their dream/nightmare, solo, peered, teamed or departmentalized.

Not all programmers recognize their abilities. Many become automatons. Do not just take for granted that others gather awareness while learning content in a given context even if they are technically learning abstraction and pattern detection. Even if you become the guru, refactor both the code and your method. Refine your toolset, workflow, process etc. The magic is the creativity applied, envisioning, detailing, and translating to a process. Iterating to refinement. Wash rinse repeat.

The "old" programmer has been the little pup, the journeyman and the top dog. Has learned humility, after indulging the ego and being the prima donna. None of "it" is throw away. You know more whys to the whats and thus are wise.

As an "old" programmer: prior to the internet you knew how to ferret out information at a library or by interview. That translated eventually to skimming books and magazines. Which proceeded later to search engines, comments threads etc. Now you run circles around most at finding the answer or the material that is vectored to one. True you used to sit with a book(s), at the computer trying stuff out for hours on days now depending on how lazy you may feel you can script something, batch it, wire up in some online app, custom local app, utility tool whatever and get results. You can do it for the onezee twozee or the millions and billions. You can use the old languages, current or latest greatest if you choose. Only you will argue with the stakeholders as to whether the time spent is worthwhile, the new programmers will not.

The decayed material knowledge and numerous cycles have given you context. You can confidently create the start up, become the director or tool.

again I enjoyed this,

another "old" programmer

Alexander Gabriel **October 11, 2016**

28 year old here. Loved to read the article/post and all the comments. In fact, loved reading it so much that I actually leave a comment - For the first time ever. Why? Because guys like me (I assume I am not the only one) admire and love to work with &#65533;oldies&#65533; like you.

In my opinion, all of you reached a level most of my generation never will. Why? Because you know what's happening behind the curtains. You&#65533;ve been there. You've taken care of &#65533;your own memory&#65533;. You understand how many cycles an operation takes. You know your sh**! Then there is people like me: Blindly jumping on to the next hype-train; Thinking we are all that great because we mastered the latest framework within a week; Yet we fail to answer the simplest question about a programming language, its fundamentals, design patterns or memory management.

If I had to choose between working with you and a team of people working on the next "unicorn startup", I would choose working with you over and over again. Nothing beats the experience and knowledge you've acquired!

Disclaimer: This is only a small excerpt of how I feel about it. I could probably have really nice, lengthy conversation about it over a coffee.

Chris **October 12, 2016**
Alexander Gabriel wrote "...taken care of your own memory..."

I would like you to clone yourself and send your clones to be my programming students please.

Ben **October 12, 2016**
Thanks again to all who left such great comments. There are so many interesting threads of thought, I don't know where to start. And honestly, I don't know if there's much I should/could add.

I think @Tom Harrison has a great perspective, which is to embrace the short half-life - it's part of what makes our jobs unique and interesting. I appreciated that thought.

Also, I have to say, I'm inspired by all the programmers who can more legitimately call themselves "old" if they wanted to :) - thanks so much for sharing your perspectives. Fun to hear about your experiences. @Gary, @Robert, @Marty just to name a few.

Kasper **October 13, 2016**
Hi Ben

Congratulations on becoming 40. If you didn't, you were already dead! So yeah, don't feel bad about it.

Your knowledge does not half in 10 years.

Concerning experience, I too have many years under the belt. But I don't feel lost the same way you do. I never really made it to the front end. I know a bit of JQuery and that's it. My previous company I did very little gui, most was auto generated, and it was a bespoke thing. Before that I did in my spare time some Python, Smalltalk, Java and heck some IBM Gui programming I have long forgotten the name of. Since I have only witness the javascript bewilderings from the sideline, I am only amused that I did not jump wagon every week for another framework. That I did not buy into the whole NPM package model gunk where there is no control whatsoever and so many packages under 10 lines of code. It was bound to go wrong and it did at some point where Node couldn't build.

I feel that 90% of what I know I can carry over. Most gui is about events originating from users doing things. That is about the same in most frameworks, that core concept. A few years back Microsoft announced MVVM over MVC. I really didn't feel too bad about that either, because I've always felt that MVC did not make sense (and a view model did). Whenever I read about

the next big thing mostly I can track down that feature to smalltalk (invented in the 1960's), so rarely something really new comes along.

Finally, there is a big shift in the speed of the internet and the availability of information... and the IDE's. When I started doing Java around 1998, you frequently had to turn to the web javadocs to understand the arguments to call a method. Being good at memorizing bullshit like that made you efficient compared to your peers. Today with the IDE etc. you can look most things up within the IDE of using 10 secs and a search engine. Hence it is the non-trivial things nowadays that are in focus. And those complexities do not turn at the same speed. Far far from it.

I try to blog on relevant programming topics on http://firstclassthoughts.co.uk/ give it a shot ;)

cheers

Andrey **October 13, 2016**

Some time ago I felt like new knowledge appears every year and there are a lot of stuff to learn. Anyway, as a full stack developer I rely on
- SQL. This will not die for decades.
- Linux basic commands(or may be people call this posix), ack, grep, rsync, bash scripts. I bet that is for decades too.
- Algorithms, O(N), cache awareness, etc. Seems not to change a lot and mostly used indirectly.
- Server side framework. For example, rails or sinatra or nodejs based. This changes every few years, while basic ideas are still the same.
- Javascript and client side frameworks. That changes really fast, although basic and not so basic knowledge of DOM, CSS, events, jQuery is more or less stable.

Overall, it is possible to learn important blocks and glance at a latest framework

MarvinMartian **October 15, 2016**

40??
In 2 years I will be 70. I'm better at programming now than I was when I started. Fortran and Cobol. The "language" to learn is 'C' everything else is easy after that. Learn it on a small platform.
I get paid to play. I don't plan on EVER quitting.
Have nice day! - attribute to FPSRussia

michael l parks **October 27, 2016**

awesome post... great perspective... I believe you are spot on...

Anders **November 02, 2016**

This really resonates with me. After 35+ years of developing for fun, out of which 20 I have done development professionally, I have the same feelings towards everything new as you do.

Stanislav Jogui **November 03, 2016**
Thanks a lot for your sum up. Being 40 as well I totally agree with it.

Dino **December 08, 2016**
Spot on Ben spot on

Allie Laine **August 21, 2017**
Really very interesting post. Thanks for sharing

Tom **September 12, 2017**
Hi Ben,

You have written this post nicely.Thank you for sharing.

Robert **March 19, 2019**
Hi Ben,

My father is 66 and he says "I still carry on" ;)
Awesome post.

SENLA **March 22, 2019**
I tend to agree, but the total image of all the knowledge gained makes me a better programmer, even what will become "crap" in a few years, is valuable experience in thinking.

Zivoke **April 30, 2019**
Hi, thanks for sharing. You spoke same thing of my core. Good point about SQL and stay relevant.

Cloudanalogy **June 20, 2019**
good post to read keep the good work thanks,cloudanalogy