# Software by the Foot

## April 26, 2003

A couple of weeks ago, I got an e-mail from a friend. It was Joanne, one of my college buddies, and she was announcing her retirement from professional computing – as she would say, after 30 years of computing and 26 years of making a living at it.

I've heard this before – from Joel, another of my college buddies. It was a couple of years ago, and it wasn't too much of a surprise. Joel always wanted to be a hands-on dad, and he'd been commuting between Colorado and the Bay Area for a while. Besides, his wife, Jean, is employed in "the business" and makes decent money. I figured that Joel's case was an aberration – fatherus extremus, the stay-at-home-dad.

My initial reaction to Joanne's announcement was "Wimp – pooping out. Too bad, she was really good." But that didn't sit right. Thinking further, it occurred to me that no matter how I count it, 26 or 30 years is a long time … a whole career! Still, to a lifer like me, her departure seemed premature.

Now this isn't really about Joanne. Joanne's just the catalyst. The lease on my office is coming up for renewal. My lease was signed ten years ago when office prices were cheap. And, well, office prices aren't what they used to be … and neither are software consulting revenues. One's way up, and the other's way down. So, no matter how you cut it, something's got to change.

Actually, this story started about five years ago while moving a bunch of stuff. My (then) little friend Leigh Ann was helping out, lugging piles of stuff from one place to another. Tired and a little confused, she eventually noticed that lots of this stuff was really old. I was the adult, though, and I told her to keep moving it, and that's what she did. And for five years, I've been watching my old stuff get slowly older.

Given the timeliness of the lease issue, we're now getting down to brass tacks – money. I can keep all of my old stuff and move to new large digs, or I can take just what I need to more modest digs. Big digs are big bucks, and small digs are less. I'm good at math – some of this stuff needed to go.

Taking a look around, I saw program listings and tapes from the 70s, epic memos and giant floppy disks from the 80s, and manuals and floppies from the 90s. There were two full file cabinets, six banks of shelves, two desks with all of their drawers, a whole storage room, and ten banker's boxes full of old files. Taking it all in, I knew I couldn't bring myself to tell the next Leigh Ann to move it all … not twice.

So, what lives and what dies?

Fortunately, my office is well organized: a shelf for DOS, a shelf for Windows, shelves for development tools, shelves for current applications, shelves for old applications, and

shelves for applications so old that I'll never need them again – the later shelf was organized five years ago, and what was true then is very ripe now. I started there … two weeks ago.

The drill was simple. Take apart the packaging, save the floppy disks (hey, they can be written over!), put the paper products into the recycling bin, trash the leftovers, and move on to the next package. Sure, this was time consuming, but saving trees is worth the trouble.

There were FAX programs, backup programs, editors, compilers, information managers, database managers, debuggers, communications programs, word processors, spreadsheets, project managers, money managers, and stupid little utilities whose peculiar function I can *still* remember.

After tearing into a couple of packages, I got to thinking about what was tearing through. Imagine this:

- A world where software arrived via UPS or Fedex.
- It came in a box.
- With lots of floppy disks.
- And a manual, maybe even a hard-bound manual

Each package represented the life's work of a small team of young programmers passionately dedicated to setting the world afire with their incredible-new-idea-never-before-thought-of-by-any-other-person-in-all-of-human-history. Of course, along with the programmers came an entourage of documenters, project managers, and quality assurance engineers, all fanatically dedicated to achieving a crescendo called "the release." Inside each package were colossal egos, epic all-nighters, benders lasting for months, birthdays and Mother's Days missed, pizza breakfasts, inevitable flameouts, and manifest destiny.

These packages were often solutions to problems that didn't even exist until the software itself came along to define them. The test of a prophet was how quickly he could understand the meaning and function of each new class of software as it sprung from the ether. Collectively, these teams were redefining the world and how it worked.

There were five FAX programs alone: WinFax, UltraFax, WinFax v2, WinFax v3, and WinFax v7. As I dismantled each package, I saw a progression of killer ideas turn into businesses, with upgrades, then maintenance plans, and finally obsolescence. After the FAX programs, the paper stack was three feet tall, and the floppy stack was a foot high.

Next, came the editors and word processors: Brief, CodeWrite v3, CodeWrite v4, CodeWrite v5, Word v2.0, Word v2.0a, Word v2.0b, Word v2.0c, Word v6.0, etc, etc, etc. As I did the dismantle dance, I reflected further on the progression of this software over time. In the beginning, there was good software which got (mostly) better with each successive release. Eventually, the real crap began to take over – as feature heavy as it

was bug-laden … like a weed genetically engineered to completely decimate the itinerant color. Of course, I'm talking about the Microsoft products – Word 6.0 was actually released with 30,000 bugs! (It took years for us traditional programmers to figure out what Microsoft had done to us … more on that later.)

The paper stack had grown – there were three of them now. The floppy stack had also grown into three, each in danger of falling over, and with the overflow having already filled the floppy boxes in my storeroom.

Stepping back to survey the scene, I could see some trends:

- Going through with this culling would yield about 100 feet of software … *software by the foot*
- Shortly, there would be far more floppies in the storeroom than I could ever use … *ever*
- This culling could not be done in a day, at least not if I wanted to maintain due deliberation

*A hundred feet of software.*

Before my eyes lay the fossils of an epoch that burned bright, and has nearly burned out.

It took 10 days to get through the shelves, the filing cabinets, the desks, the banker's boxes, and the storage room – an hour each day, or until my recycling bin filled up, whichever came first. On the 10th day, wanting simply to get to the end, I borrowed a large wheel-away recycle bin and got serious. By then, I was getting good at chucking stuff … it took only two hours to go through a couple of hundred pounds … the last 25 feet.

Along the way, I found a collection of Daytimers from 1987 through 2001 – daily evidence of how hard I had worked for 14 of my 33 years.

I found hoards of memos, proposals, project plans, design documents, delivery documents, (very fat) test plans, bug reports, bug report responses, contracts, and user documents. All were elegantly and meticulously done … personal statements of who we were and what we were about. We were professionals, doing our best work in a field we loved.

At one point, rhythmically thumbing through the stacks, I was struck by their quality and substance. For a single second, I was transported back to those times, and I was flushed with a wave of pride and awe. A single thought: "I was good, I was *very good*." *Was*?? Interesting tense … I promised myself to come back to this topic later.

I found other things, too. Reams of international character sets, mainframe communications protocols, mainframe communications manuals, seminal works in Windows user interface design, entire program development systems, all obsolete and extinct. All of the documents were fat and brimming with evidence of marathon efforts

by smart, thorough, and diligent people and organizations. The organizations themselves are as extinct as their software (though the companies still exist) – they've been downsized, inverted, retargeted, and streamlined. But what about those great people who did such wonderful work? By now, they've moved up and out, having been replaced by younger, greener meat. In the end, the formal processes that produced all of this software simply don't exist in corporate America anymore, much less anywhere else. (More on this later, too.)

And I found even more. Product brochures for long forgotten products from long dead companies, files full of long-irrelevant problems and solutions, phone bills for hundreds of dollars, Fedex bills for daily software deliveries, and expense receipts for my many weekly trips to various customer sites – all evidence of not only my efforts, but the efforts of thousands of engineers in hundreds of companies to produce hundreds of products – all now but dust in the wind. Not even the engineers themselves remember these products. And I remembered the products that *I* had made, and forgotten. Instead of hoisting a Coke to the past, I got a sinking feeling of desolation and futility, noting that essentially all of the hard and brilliant work of a generation was not only gone, but without even the trace of representation in any current products.

Given all that I trashed, what, you might ask, survived the purge? Certainly, the software I currently use survived. Even this software bears ghosts, mainly because it is almost all old. If this software is evidence of a functioning business, I thought, the business must surely be living in the past and probably on fumes. But no – the *most* recent software I use has no representation on the shelves. It's soft software – delivered via Internet directly to a hard disk. And when it's obsolete, it will simply be blasted out of existence with a single mouse click, leaving not even footprints on a shelf as testimony to the aspirations and hard work of its authors – essentially as if it never existed at all.

So what, then, is the sense of a generation's worth of toil … what did it all come to? Judging from my pile-o-software, one is tempted to say "nothing … it came to nothing". After all, fully 98% of everything we ever produced amounted to naught. But that's grossly unjust and oversimplified. At the very least, one could claim that all of this effort and investment allowed our industry to progress to the state we find today. That would be true, but insufficient.

Mulling this over in the shower (where my only really good thinking ever occurs), I saw a different analogy: a doctor to her patients. Consider that a doctor works tirelessly to enrich and save her patient's lives, only to have them all die in the end. Could I say that the doctor's life and good work were a waste? Obviously not.

In fact, the valuable products that we've all produced delivered their benefits immediately to their consumers, and therein their values were immediately realized, and we were immediately paid. And neither the product nor the compensation are immortal, and neither are we. Ahh, the way of the world!

But along the way, we changed the world in ways few groups of people ever have. And though few people will ever remember us or know how we did it, we will remember if we

choose to and are able. All of this software was the embodiment of epic efforts and epic changes. And despite the demise (or constructive demise) of the organizations we've worked for, we survive as individuals – epic educations and all.

So, after all this, I can't begrudge Joanne her choice to do something different – life's short. I choose to continue to do what comes naturally – life's short.

## Part II

OK. We were programmers.

But nurses were nurses, teachers were teachers, soldiers were soldiers, and doctors were doctors. Each has their own epic story, and most grew up along the way – just like we programmers did. Each story has great lessons that transcend individuals and illuminate fundamental truths about business and society.

IN THE BEGINNING, there was fun. We played with computers because they were fun, and it was fun to make them do interesting and useful things. When we started in the early 70s, there was no money in computers. They were primitive, and anything we could make them do certainly beat anything they were already doing. Access to computers was rare and privileged, and anyone with the stink of a computer on them was either a bizarre nerd, a god, or both. We were our own little clique, speaking our own language, defining our own problems, and exulting only within the company of our peers.

As with all cliques, hierarchies formed – the best were gods amongst nerds, and the worst were nerds amongst gods. The main criteria for judgement: art. The name of the game was to produce a reliable program by encoding the most germane and poignant abstractions in a computer language in the most efficient and elegant way.

Say what??

Think "Shakespeare". What distinguished his works from all those writers whose works were forgotten even before *they* were? Two things: Shakespeare chose profound and universal topics, and when he wrote he created imagery through brilliant turns of words and phrases that together created timeless worlds.

So, great programming was about divining the essence of a problem, and then putting it into a computer language in a way that did credit to the problem and the programming language at the same time.

Art.

Nerd passion, but still passion.

Furthermore, the results were spectacular. We were there when computers first became useful in the lives of average people. We conceived and programmed the first games, accounting systems, financial trading systems, file systems, word processors, communications programs, networks, and all manner of supporting technologies.

Well, almost. The generation immediately before us actually created the first versions of some of these programs, but they were working with mainframes – behind thick glass doors, walking on expensive false floors, in carefully climate controlled rooms, in large data centers funded by and for large corporations. Society regarded anyone allowed to touch or affect these machines as the priests of the new, mechanized future. Additionally, society saw computers as something apart from itself – more of an adversary than a help.

Our programs were aimed toward smaller machines (first minicomputers, then microcomputers, and eventually PCs) that directly affected many more people, essentially empowering them and freeing them from mainframe slavery. The importance of our results, combined with the turf on which we operated, conferred the mantle of the priesthood on us, too. Liberating people from the tyranny of the mainframe made us heroes of a sort. We were just kids – wonderkids – learning that we could thumb our noses at world order in important and relevant ways.

So, why did we become programmers?

The fun, the art, the thrill, and the respectability.

And because no one told us we couldn't.

This is how amazing things happen.

We were programmers.

## Part III

In the late 70s, I had a lab at UC San Diego.

One day, a rangy old guy (25 years old or so) took up residence on a large slab of foam in a far corner of my lab. He slept there. After a couple of days, he came up to me and asked me for a floppy disk containing my latest work. The rangy guy turned out to be Bill Atkinson, one of the first employees of a company I'd never heard of: Apple Computer. Under instructions from my boss (and against my better judgement), I handed over the goods: both the current release and the release I was working on. Shortly after, the current release came out as Apple Pascal v2.0, and the work in progress came out as v2.1.

Of course, Apple went on to become the most famous garage startup of all time, partly on the strength of our system. Equally important, our society was changed because of the computing power that Apple brought to everyone's desk and counter top.

In truth, my work was a sliver of an increment on a system that Roger Sumner had built. And to run, it required a substantial program written by Mark Allen and my roommate, Rich Gleaves, and that was based on another program written by another roommate, Joel McCormack. And all of that work depended on and was supported by about 30 other students, mostly undergraduates. *Undergraduates*.

Some went on to become early Apple employees. All helped change the world, through Apple Pascal and in other adventures.

Eventually, this project was booted out of UCSD and into the private sector – it had been a worldwide success and was making an embarrassing amount of money for the erstwhile non-profit University of California.

Our values remained consistent: produce the most useful, elegant, and bug-free system we could. We had faith that this would be sufficient to allow our system continued success.

We were wrong, and it took years for us to figure out why.

In 1982, IBM created the IBM PC. They ran an informal competition to determine which software would be its flagship system. There was our system (UCSD Pascal), Digital Research's CP/M, and Microsoft's MS-DOS.

At the time, Microsoft was 10-or-so guys we'd never heard of … spread between New Mexico and Washington.

We and Digital Resource competed on our technical merits, at the collegial pace of our industry at that time. Microsoft competed on promises, work carried on at breakneck speed, and unusual business savvy.

Microsoft won.

None of us knew it at the time, but this was the seed of a tectonic shift in our tiny industry. A new and dominant metric for success had been created: money and power.

As we plied our trade in subsequent years, some of us adopted MS-DOS as our platform, others adopted UNIX, others worked on projects where the platform wasn't relevant, and still others dropped out of the business. For those that remained, the prime directive continued to be quality – as if there was a programming guild, and we all had a responsibility to it.

Meanwhile, Microsoft grew and got stronger (– the understatement of the century!). In the late 80s and early 90s, Microsoft was busy creating Windows at the same time as working with IBM towards creating its successor, OS/2. Furthermore, it was also developing Word and Excel, which would later become Microsoft Office. As we all know, Microsoft and IBM got a divorce, IBM got OS/2, Microsoft got Windows and Windows NT, and Office eventually killed off its competitors and ruled the world. The implications of this shakeout, though, were horrendous.

First, Windows NT eventually killed OS/2. Why? Because OS/2 was a technology statement – the embodiment of the best technology available at the time, and different than anything that had come before. Windows NT was also a technology statement, but carefully engineered to be substantially the same as regular Windows. In fact, in this case, *the inferior technology won!*

Second, when Word v6.0 came out it had 30,000 known bugs. As good programmers, we all believed that an acceptable release had no bugs. And failing that, maybe 1 … or 2 … or 10. But not 30,000! Shockingly, there was eventually a Word v6.0a, v6.0b, v6.0c, and v6.0e all of which attempted to fix Word's bugs and Word's bug fixes. Yet, Word prevailed over its rivals largely on the strength of its long checklist of features (working or not) and Microsoft's arm-breaking marketing genius. In fact, in this case, *the inferior product won!*

And third, we all knew that in creating any of its products, Microsoft had invented nothing – it harvested the low hanging fruit off the grounds of universities and generally floating around the industry.

It took us years to understand that Microsoft isn't, wasn't, and never will be in the business of making excellent programs. They had changed the game so that it was about money.

Money and power.

## Part IV

One day, the Internet hit.

Life got much better, and much worse.

Many of us programmers were out of position, as Microsoft nearly was.

For the programmers, the Internet represented either another technology to eventually master, or an exciting new technological frontier to be grabbed onto with both hands and ridden to the stars. The perspective depended on the programmer, of course.

For Microsoft, it represented a monstrous competitive threat because it was a colossal technology that Microsoft didn't already own – its money and power were threatened. Microsoft's reaction was to throw amazing money and coercion into our industry, far beyond the debacle we saw with Word.

The dot-com explosion is still fresh in our minds, and we're only now able to understand the damage it did to our field's values.

In as much as Microsoft and the business community had made computing about money (and rightly so, in many ways), *everyone* followed this lead by jumping onto the Internet gold rush. I mean *everyone*. With so much money floating about, anyone who could move a mouse or peck on a keyboard stood up to stake a claim on even the merest fraction of it. This meant that in place of a cadre of seasoned professionals proceeding judiciously and responsibly, hoards of arrogant, ignorant, and greedy newbies flooded the streets, claiming deep expertise in technologies not even fully hatched. Predictably, university enrollment in computer programs exploded.

Luckily for the hoards, none of their programs were ever really tested – as quickly as their programs were written (or at least started), their new companies were purchased for ungodly prices, and their programs were discarded.

Meanwhile, as the hoards were making trillions in option dollars, the old-timers sat dumbfounded, transfixed, and demoralized. We suspected that the options were all illusions and that the newbie systems weren't worth the computers they ran on. At the same time, though, we had that sinking feeling that the programming physics we all knew *could* be defied, that somehow those dot-com millionaires were the brilliant and deserving ones, and that we old-timers were all irredeemably stupid and worthless relics.

Just as cash had no place in a world of stock options, we apparently had no place in the new technology order.

Power and Money and Greed.

## Part V

The day the music died wasn't much better, though.

If the dot-com crash ended the outrageous claims of the hoards, it left a different problem: an oversupply of programmers and an undersupply of work – work of *any* kind at all.

Good programmers have been out of work for a year and longer. New technologies that would have caught fire three years ago are languishing, and some may not ever take off.

And the hoards are deserting the ship … claiming expertise in whatever other fields they can, but generally making no more money than they're missing in the computer field.

And university enrollment in computer programs has steeply declined.

All things being equal, this should be good news. Given our relative poverty, I'm happy to say that the only people remaining in our field are the ones that enjoy computing for its own stake.

This would be a very bright time were it not for two important events that popped up in the early 80s and has compounded inexorably ever since: FASB 1984 and Globalization.

FASB is the Financial Accounting Standards Board, an accounting group that deliberates on significant accounting issues, then makes pronouncements that lead to Generally Accepted Accounting Principals (GAAP). Essentially, the pronouncements of FASB determine how companies allocate, use, and report money.

I remember a particular panic that occurred amongst corporate management in 1984 after FASB made a ruling on the accounting of money spent on computer program development projects. Basically, they pronounced that these sorts of projects have two phases: the development and the implementation. The development phase involves the

hatching of an idea, the research to determine whether the idea is viable, and the work done to structure the implementation phase. The implementation phase involves the actual programming, testing, and delivery of the resulting computer program. FASB pronounced that the development phase should be expensed, and the implementation phase should be capitalized.

In 1984, my priorities were clear: serve my client's programming needs as best as I could. I neither understood nor cared about an abstract accounting principal, though I suspected that some day I might.

In 2003, my appreciation for this principal is profound. Basically, one expenses the development phase because it's speculative – it may or may not lead to actual revenue … just like taking a prospective client out to lunch. One capitalizes the implementation phase because it's a sure thing that it will finish (relatively) on time, will have a (relatively) predictable cost, and will have a (relatively) predictable lifetime – kind of like buying a tractor … one writes it off over a number of years because it will be useful over a number of years.

The end result is that according to FASB, once a product is defined, creating it is essentially like turning a crank – you do it and the product pops out. The nasty implication is that anyone can do it – no special talent or significant art is required. As programming technology and tools have improved over the years, this has become more and more true.

And accordingly, the more of an oldster I become, the less economically suited I am for the programming activity I love.

Nasty!

Furthermore, globalization is beginning to take a horrific toll. Globalization is basically the free flow of money and resources to the countries that can employ them most efficiently. It has already ravaged the American consumer electronics industry, and it is getting ready to do the same for the American programming industry. For example, a programmer in India costs $20/hour to do the same job that many of my peers charged $100/hour to do three years ago. American business *must* export its programming work overseas because its competition is already doing that, or will do it soon.

In fact, it's far more efficient for an American company to run a programming project in America. There's no language barrier, no time zone barrier, and no issues regarding the ownership of the resulting program. Regardless, models exist and are under development which combine the advantages of American management with foreign labor. As foreign labor becomes more educated and capable, models will arise which eliminate more and more of even the American management component.

In short, ideas flow, money follows, and lifestyles follow quickly behind.

Very Nasty!

As testimony to this point, the enabling structural transformations in American businesses have already occurred. Last month, I had the shocking experience of trying to talk with the General Manager of the local IBM office. In fact, it was surprisingly hard to get the phone number of this office, and once I did, the person answering the office's phone didn't have a clue as to how to direct me. It turns out that the functions of this IBM office (and many others) have been distributed to mobile workers spread all over the country, having no particular affiliation with a physical office except to be able to reserve a physical seat when they need to alight for a conference or quiet time. IBM very much exists, but it doesn't exist in any particular place … and it flows to wherever it can produce the largest profit for IBM.

Extremely Nasty!

## Part VI

Recently, one industry reporter declared the Microsoft operating system (Windows) to be complete.

On its face, this is an outrageous claim … kind of like declaring that everything that can be invented has already been invented.

In reality, though, this may be essentially true. Over the last fifteen years, Windows has accumulated all of the features and designs floating around the universities and available as common knowledge. And when we create applications, we naturally conceive of features and capabilities that fit within our concept of an operating system, whether or not Windows lives up to the concept. Thus, the reporter may be correct in asserting that the Windows operating system now matches the capabilities needed by the likely applications … or is 99.4% there.

Similarly, the tools we use to develop programs are in the same shape: compilers, editors, debuggers, analyzers, and so on.

Thirty years ago, none of this was the case, and people like me had vast horizons of unfilled potential in the operating system and tools market. No longer.

Microsoft's operating system isn't the only system in the world, but the other relevant systems are in pretty much the same shape.

So, then, whither the more experienced programmers?

Many of my peers haven't been able to find jobs for several years, and they blame age discrimination. Fortunately, as the owner of my own company, I'm the master of my business model, and am not susceptible to the age discrimination effect.

There are only two ways out of the systems box: become an applications programmer or develop whole new kinds of systems.

(Generally, there are two computer programming camps: the systems people and the applications people. The systems people create and develop the operating system and tools, while the applications people have expertise in a particular industry (e.g., insurance, banking, tree trimming, accounting, etc) and write programs for their industry. As honorable and revered as systems programming has been, the challenge has shifted to applications, as the ambitions of users they serve hit new heights. Examples of computationally challenging applications involve decoding the human genome, weather prediction, electronic commerce, and traffic routing and control.)

Fortunately, I have deep experience in both systems and some applications – I can leverage systems disciplines into applications work. And I'm definitely up for creating new applications that can be implemented by my friends and me without resorting to exporting the interesting work elsewhere.

But what about the systems programmers who have no more systems to program?

And what about the applications programmers whose job has been exported to provinces whose names they can't pronounce?

And ultimately, what about American leadership in computer technology?

But we do what we do because it's our calling, not because it keeps us rich.

I hope I never have to find out whether life can be fulfilling without getting to do what comes most naturally. So far, the majority of the programmer class has avoided that fate … we've been lucky, but no kind of luck keeps up forever.

## Part VII

So, what *does* it take for a programmer to be successful in the new world?

To date, the story of the computer industry has been congruent with the great American ideal: it was created by lone visionaries pursuing their singular visions against the odds, bucking convention, and risking all for the love of their craft and the elusive brass ring.

One lesson the lone programmer eventually learns is that time works against this ideal: as time goes on, "one man projects" can no longer be done by one man – the programming ecology is far more complicated than ever before. Additionally, as time goes on, all of the projects that *can* realistically be done by one man get grabbed up by one (or several) of the ever increasing ranks of smart, talented, and motivated newbies. For the experienced programmer, this leaves only the more complicated projects. Complicated projects require collaboration – either with other programmers, or with team members of other disciplines.

Of course, this trend has been underway since almost the dawn of the industry, but at the beginning of the 21st century, the *trend* has become the *rule*.

Not surprisingly, then, team management techniques have become as important as any technology, and more and more mind share is being devoted to defining and optimizing team performance.

Ironically, one of the great interests and competencies of my buddy Joanne has been exactly these kinds of team management issues. On the surface, it's puzzling that she should choose now, of all times, to bow out.

I suspect that the resolution lies within the very importance that computing has attained both in operating a company and within products sold by companies. As computing becomes as much of a business competence as a technology, senior computing professionals find that their immediate team members consist more of business-minded individuals than the traditional tech-heads. Inevitably, this lands us on far more evolved political playing fields than we would encounter just working with our computing peers – essentially getting further and further from the technology fun, and running ever harder in the rat race, with less preparation or temperament than the other rats.

I wish Joanne the very best of luck in her new environmental engineering studies, and if this was her perspective, I don't blame her in the least.

## Epilog

So, what's a programmer to do?

I'm sticking by my guns, for better, not worse. I will leverage my systems-oriented background in creating reliable and complex applications as both a singular technologist and as a member of a multidisciplinary team.

Since the world is no longer the target-rich environment it once was, maintaining my earning power requires that I station myself firmly at the cutting edges of technology and application development, providing prospective, leadership, and services that simply can't be outsourced elsewhere.

On one side of the competitive equation, there are hoards of bright, energetic nerds pushing the state of this art in all directions, and nearly all directions are important to nearly all applications of any significant value. On another side, my head is full of information collected and perspective earned over 30+ years – much of it is golden, and most of the rest is useless baggage.

Certainly, I have a great aptitude and enthusiasm for computing. During most of my 30+ years, I've worked very hard and the results have been consistently excellent. Regardless, I wouldn't describe myself as consistently brilliant – not like Joel, Roger, and others I have worked with. And now, there are many more truly brilliant people in this field than ever before.

Given that these people are the ones defining the directions I must follow, can I follow their work and apply it quickly and effectively enough to justify the lifestyle I hope to maintain?

Can I collaborate with them to make them even more effective, thereby increasing my own future value?

Is the information and perspective in my head as valuable as I hope it is, or will it simply retard my progress even as I run harder and harder to keep up with evolving technologies and projects?

Can I find something that needs doing, which my friends and I can have fun doing, which pays a decent wage, and can lead to other great projects?

Have I fallen so far off the "leading edge" that I'm really at the edge of an abyss, but just don't know it yet?

There is no way to know the answers to these questions, and there never has been.

In fact, *not* knowing these answers ahead of time is what allows us to give our best efforts in the hopes of continuing to accomplish great things. It has always been thus, and giving our best has always been what has allowed us to realize our dreams – even if we don't recognize our dreams until after the fact.

Regardless, I'll probably be moving to a smaller office at the end of the summer – lower cost structure reduces anxiety and increases survivability under adverse economic conditions. And to quote an old Al Stewart song, "it sharpens your perspective when your back's against the wall."

Our field has essentially succeeded to the point of commoditization … essentially, cranking out *software by the foot*. Maybe Carl Marx was right – perhaps capitalism (through globalization) has created the seeds of our own destruction. Or maybe it's time for us to reinvent our field, lest it actually become as cut and dried as FASB might have us believe.

Party on!

May 31, 2002