Laws of Immutable Software

Yes, yes, I know. The theme for this issue is "Immutable Laws of Software Development." Not quite the same as my title. But then, I was struggling for the right topic to write about, and a black lab and a breadbox intervened.

See, we have Molly, a slightly brain-damaged black lab, which my wife rescued back in 2007 before she met me (or, as she oddly refers to it, "the good old days"). Molly had distemper as a puppy. The SPCA wanted to put her down. My wife, having already fallen in love with her, was against the idea. Against all odds (and with much help from the vet), Molly survived, with few physical side effects. Mentally, her brain is permanently stuck in puppy mode. Molly feels that anything on the floor is legally hers, and that anything within reach on the kitchen counter counts as "on the floor." Leave a loaf of bread sitting out, and it disappears with amazing rapidity. To keep Molly at bay, my wife and I decided to order two rather large breadboxes. When the breadboxes arrived, one was broken.

I called the company and spoke to a very nice customer representative, who quickly apologized, ordered us a replacement, and simply asked us to carefully discard the damaged item—no need to return it. I was chatting with the customer service representative while she was completing the process. She apologized twice to me for the amount of time it was taking, and mentioned, "you have no idea how old this computer system is!" I laughed, told her what I did for a living, and laughingly said, "Are you still running Windows XP?" She laughed back, and replied, "Would you believe MS-DOS?"

I thought she was kidding. Nope. They boot Windows XP, which runs a driver that apparently maps a database of several FAT32 file systems into a set of virtual FAT6 files, and then use command.com to open a MS-DOS window and run a batch file to load a program that was written back in the early 1990s. And, to quote the customer service representative, "it works just fine. It meets our needs."

In February 2004, my friend and colleague Theron Leishman and I published a Backtalk column entitled "Laws of Software Motion." [1] In that column, we discovered several laws of software development by comparing them to Newton's "Laws of Motion." Newton's first law is that "An object in uniform non-accelerated motion (or at rest) will remain in the same state of motion unless an outside force acts upon it." We countered with Cook-Leishman's First Law – "Any software intensive program not given adequate force (motivation) will degrade and cease to progress."

Here is a very successful high-end cooking equipment company, with a presence both physical (world-wide) and online. They are using software that was custom-written for them over 20 years ago – and it still meets their needs! Why change to Java from gosh-knows-what? If your company's software meets your needs, and the cost of keeping it running "as it is" is less than the cost of redevelopment, well then, keep on truckin'. It is called "making a profit"! Sure, you might have to write "glue code" to keep the software working on modern hardware through the years, but it is cheaper than rewriting all the software!

To make software work for 20+ years, you have to do a lot of adaptive maintenance. In the DoD we have quite a few legacy systems that are well over 20 years old. Many of them are interactive, real-time, database-oriented, and interface with customers. If you have never taken on the task of legacy systems maintenance, it's a different world. It takes a lot (and I mean a lot) of adaptive maintenance to keep them going.

But somehow, we manage to create immutable systems in spite of the "immutable laws." B-52s still fly (and have been for 60 years). 1960s 70s, and 80s large-scale legacy systems still function. The hardware ages, the hardware gets replaced. Peripherals become obsolete, replace them with new peripherals. We have gone from tapes to floppies (8", 5 1/4", and 3 ½"), USBs, CDx, DVD, and now the cloud. And yet, the systems still work.

There is little thrill in working as hard as you can just to keep the system running, pretty much like it was running yesterday, and the month before, and the year before. It is like Alice and the Red Queen in Through the Looking Glass: "Well, in our country," said Alice, still panting a little, "you would generally get to somewhere else—if you run very fast for a long time, as we have been doing." "A slow sort of country!" said the Queen. "Now, here, you see, it takes all the running you can do, to keep in the same place."

It is relatively easy to graduate with a degree in engineering or computer science and develop applications in Java, Objective C, C#, Ruby, .Net, Pearl, or Python. Try becoming fluent in languages of yesteryear, and then transitioning to development frameworks and mindsets from over a quarter of a century ago. Having your college friends laugh at you when you tell them you still program in Jovial, Fortran, or Cobol.

Maintenance programmers, it is your turn. We do not appreciate you enough. Take a bow.

David A. Cook, Ph.D.
(and former maintenance programmer)
Stephen F. Austin State University
cookda@sfasu.edu

(Reference)

1. http://www.crosstalkonline.org/storage/issue-archives/2004/200402/200402-Cook-2.pdf.

Every author yearns to reference himself or herself sooner or later. I feel MUCH better now.