0100010001000
0101010001000
1000011010101
1101000101001
?

Your on the Street Reporter

Uyless Black

Software Complexity and Software Failures

Software Complexity and Software Failures

> For routine matters, let's go with the computer to relieve us of tedious and repetitive mental tasks. But let's be careful about replacing our cerebral computations with lines of software code. Millions of years of experience have created a mysteriously marvelous brain, one capable of fantastic intuitive actions. Try as we do, we have not been able to make computers intuitive. Yet, intuition guides many of our daily actions. We place ourselves at risk by assuming the computer knows all. It only knows what we tell it to know, often without our intuitive input to its logic. After all, how can we code intuition into software? We don't know much about it ourselves. We just know it works and that it keeps us out of harm's way.
>
> --- anon

February and March, 2010

Hello from Your on the Street Reporter. During the past few weeks, I have not been on the streets or highways. I have been involved in migrating to a new personal computer (PC), as well as a new notebook computer (NB). I will use the PC at home and the NB on the road.

I am fairly adept with computers and software, yet I have spent many hours configuring these two machines with the *same* software applications I have been using on my old PC. Here is a recount of some of my experiences.

Windows 7 on a new machine. While on the road doing *The Pacific Coast Highways* reports, and in order to begin a migration away from the abysmal Vista operating system, I purchased a Hewlett Packard (HP) notebook along with the operating system, Microsoft's Windows 7. The Windows 7 download, using a hotel-furnished wireless connection, took almost three hours. I had to use the download because the notebook does not have a CD drive.

A few days later, Microsoft's Web site informed me the (purchased) copy of Windows 7 had "expired," and I had to purchase the software.

As mentioned, I had already purchased 7 from the Microsoft Web site. I attempted to correct Microsoft's error through this site but without success. I might have been able to work directly with Microsoft's telephone help desk to prevent buying Windows 7 again, but I was occupied with traveling and writing, and did not want to spend even more hours hassling with them. With the expectation I could work out the problem later[1], I bought an external CD drive, *another* copy of Windows 7, and was finally in operation.

Windows 7 to replace Vista on an old computer, aka, you can't get there from here. After returning to my home, I had to buy yet another copy of Windows 7 (an upgrade version) to replace Vista on my desk PC. The copies I bought while on the road did not work for an upgrade. So, I bought my *third* copy of Windows 7.

[1] Which I may be able to do. The download provided my software serial number, then deleted it before I could copy it onto paper. However, it is stored. If you need it, click on My Computer, then right click, then click on Properties.

This last copy of 7 did not install itself because it informed me I had four files on my PC that were incompatible with this new operating system. They had to be removed before Windows 7 could install itself.

Seemingly, no big deal. I would use the Windows Uninstall feature to remove these pieces of software. (I had no idea of what they did in the first place.) However, upon clicking on Uninstall and looking for the files on the Uninstall window, two of them were not in the list of "uninstallable" programs. So, I entered their names into the Search Window. Microsoft informed me they could not be found.

Therefore, at this point, I could not install Windows 7 because two software files had to be removed. But Vista would not let me remove them because they did not appear in the Vista uninstall list, and they could not be found by the Vista search facility. Figure 1 is a visual depiction of the cycle I was in. The figure shows only File A. As mentioned, the installer did not like four files. (Including iTunes, which I successfully uninstalled. I wonder what might be involved in the reinstallation?)

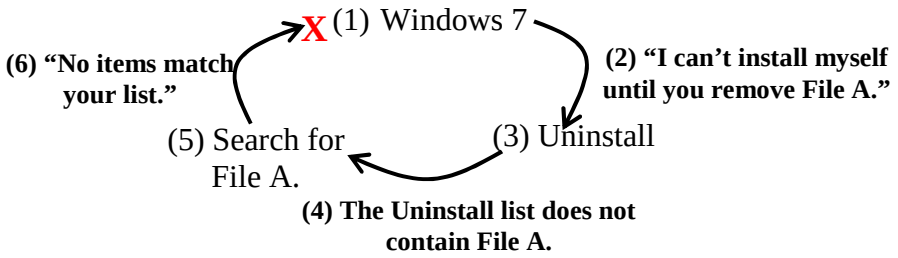


Figure 1. Path to nowhere.

I am confident a technician versed in Windows could have told me in a few sentences how to find the two "unfound" files that 7 informed me I had to uninstall. Nonetheless, it seems to me there is a huge gap in the partnership of Window's installation, uninstall, and search *software in how a PC user interacts with these programs.*

If the cycle shown in Figure 1 makes sense to you, please send me an email explaining how three critical pieces of software in the most widely used commercial operating system on earth do not interwork correctly to allow a user to load a new operating system.

A possible hardware problem. In the meantime, my old PC had stopped working. The use of the mouse either froze the system with no error message, or briefly showed a diagnostic pertaining to an Intel name and then froze. If the PC actually operated (and only in the Windows degraded "safe" mode), the four-year accumulation of thousands of programs, the doggedness of Vista, the awkwardness of how the security software interworks with other stuff translated into such slow responses that I put up the white flag. As mentioned, I purchased a new PC.

I now had a new PC and a new notebook. And because I wanted to migrate away from Vista, I also had three copies of Windows 7 (to run on two machines), a new CD external drive, new copies of several expensive applications, such as Microsoft's Office…and more.

©Uyless Black 2010

In the past, I liked to figure-out the sorts of problems described here. But now-a-days, not all that much. I've become a layman user of computers. Spending half my waking hours for three weeks configuring two computers, one printer, a CD, a router, some wireless connections, several applications, three copies of an operating system, security modules, *ad nausem* is not how I want to spend my time.

Old printer did not work with new operating system. My recently purchased notebook and PC now ran correctly with Windows 7. But other than doing a straight print job, my old printer was not compatible with 7. I use a printer for scanning and faxing. But the *old* printer did not scan and fax with 7. So, I had to buy a *new* printer. It's a fine HP machine, but….

The installation disk that came with the printer did not have the proper software to interwork with 7. I did not know this fact until I was informed during the printer installation process that I could not use the printer's full functions. Thus, I spent several hours on the phone with the HP help desk to have them download the proper software. I tried to use their "self-guided" tour to load all the needed software. Without success.

HP informed me the CD that came with the printer was not set up to work with Windows 7, only Vista and other older versions of the Windows operating system. I asked them how many thousands of hours they had spent on the same problem I had. They did not respond.

Repurchase of several applications. In addition to the expense and hassles just described, I also had to repurchase several applications, such as Microsoft Office, Adobe, FileMaker, etc. Why? Because my CD-based copies of these products are setup to *upgrade* copies on *older* operating systems that run on *existing* computers. For my two *new* computers, I had to start from scratch.

Even with these new packages, I had to work with help desks and experts in (a) WORD, (b) PowerPoint, (c) Adobe, and (d) FileMaker because the CDs I purchased did not have all the needed installation software on them. And this is just a beginning. I have yet to try to load other applications that I use less frequently.

Router and security software. I use Verizon for my broadband service and AOL as my Internet Service Provider (ISP). Verizon furnishes firewall software as part of its product, as well as virus protection facilities. I logged-on to the Verizon Web site to download this software into my new computer. No luck, I had to call their help desk.

I spoke with two levels of help desk personnel who, after taking control of my computer, could not get the security software to install. Their guru geek put me on hold for 30 minutes because he had to call the security software company. Verizon had moved away from its former security software vendor to McAfee. I like McAfee's offerings and was pleased Verizon had put it into their security product. The problem: It did not work.

After four hours and ten minutes of phone talk and hold time, McAfee took over my machine and finally got the Verizon software installed. My cordless phone ran out of battery power, and I had to resort to a phone in another room in our house to complete this call. (In our modern world, always have a backup.)

©Uyless Black 2010

Immediately after installing the Verizon software, I noticed a degradation in performance. BV (Before Verizon), I could click on one of my folders, and it opened instantaneously. Now, AV, when I click on that same folder, there is ten-to-twenty second delay before the folder opens (but sometimes no delay (?)). The only thing I added was the Verizon software. It appears this software is "looking over" my clicks before it allows the clicks to click. But I'm not sure, and the hold time to reach a Verizon expert can run into hours. I've put this task on hold.

Now what? During these times, a cryptic message began to appear on my screen that some Dell computer program "scripts" were in error. Shortly, this message appeared every time I turned on my machine, and I had no idea if it was associated with 7, WORD, or some other component.

My machine was new and relatively "clean." To get this message on the screen, interrupting an ongoing session with, say, Adobe---after spending close to two thousand dollars on clean hardware and software---was a kick in my automated ass.

I had yet another work session with a help desk. In this case, Dell. The technician informed me there was an error in one of Dell's programs dealing with the Dell Support Center. I was not interested in this software executing on my computer, but Dell runs it in a background mode to monitor operations. On these occasions, it was interfering with other applications because two lines of code were either incorrect, had been corrupted, or did not sync with some of the other newly-installed applications on the machine. I have underlined two lines of code, to which we refer later.

Dell had to uninstall and then download a new (corrected) version. This operation consumed the better part of an afternoon.

Welcome to the World of Increased Complexity

Why must a seemingly simple task of purchasing a new computer and installing software become such an arduous task?[2] The best answer I can give to this question is: The reason is due to the volume and complexity of software that is installed on modern computers.

How much software? How complex? To answer those questions, and hoping not to become too technical, let's take a brief look at Toyota, the company of accelerating engine fame.

Toyota Troubles

Toyota is in the headlines. I am reading the articles closely because I own two Toyotas. I'm informed my cars are not part of the "unintended acceleration" (aka, "unintended funeral costs") problem. Maybe so, but millions of other people drive Toyotas. I am not worried that my Toyota might plow into another Toyota and its driver. It's occurred to me that another Toyota might plow into my Toyota and me.

Nonetheless, I continue to sleep well at night because I have come to accept the world I live in. One in which I have no control over its complexity. Try as I may to make life simple, with each

[2] I don't know how non-technical people manage to do it. As you see, I had many hours of technical assistance, and I'm a semi-geek.

passing day my world becomes more complex. But it's not my world *per se*, not my confined personal universe.

It's the external world with its associated complexities that have made their way into my space, into my once confined universe of relative simplicity. This encroachment has sometimes been welcomed, because it has made my life more pleasant and productive. But the remainder of this essay will take the tack that complexity in man-made systems is cumbersome and sometimes dangerous. It is likely to become more cumbersome and increasingly dangerous.

Here are some quotes from an automotive expert named Mike Allen. I pulled his comments off the Web:

> Toyota has recalled millions of cars and trucks—4.2 million to replace floor mats that might impede throttle-pedal travel, and 2.4 million to install a shim behind the electronic pedal assembly. All of the affected pedal assemblies were made by Canadian supplier CTS. Toyota's boffins have documented a problem that can make a few of these pedals slow to return, and maybe even stick down. Problem solved.
>
> But the media, Congress—and personal-injury lawyers—smell the blood in the water. Not to diminish the injuries and a few deaths attributable to these very real mechanical problems, but they're statistically only a very small blip, which may explain why Toyota took so long to identify the issue, especially when it has symptoms similar to the similarly documented floor mat recall. Plus, sudden unintended acceleration (SUA) is notoriously difficult to diagnose because, more often than not, the problem can't be repeated in front of a mechanic.

I agree with Mr. Allen that personal injury lawyers and many media sharks make their living only if someone is seriously bleeding. But I disagree with most of his other observations. I'll explain why shortly.

Software Complexity

In former days, simple mechanics controlled cars and airplanes. Now, it is complex software. It dictates much of our daily lives; from our chat sessions on the Internet, to our ATM bank sessions…to airplanes landing safely.

Software is as integral to our lives as waking up in the morning. Try getting through any part of your day without software guiding you through your thoughts and movements. Unless you live in a cave and take your water and food from your own private pond and pasture, you can't do it.

Yes, software, the esoteric and abstract piece of mystery that is as common to our functioning as brushing our teeth. Yet, many people do not know what software is. For those readers who do not know, take a look at the box titled Software Primer on the next page. Others can read on.

Software Primer.

Software takes the visual form of a computer language. Thus, it is written in somewhat readable English, such as: ***If A and B are equal, Print "A and B are equal." Otherwise, Print "A and B are not equal."*** This sentence, written by a software programmer, is translated by the computer into binary (1s and 0s) instructions which dictate the actions of the computer and the printer.

Thousands of these statements (called lines of code) may be written to make up what is called a computer program. In a sense, these instructions are much like our everyday logic. For example, we might apply pressure to a lever to change the flaps on an airplane wing, which would result in cables moving to make the alteration. With software, we apply this pressure to a lever, but it is connected to computer and software instead of the cables. These automated components take over and direct the actions of the cable.

Software is potent because it can be written to make decisions on behalf of humans, as shown in the Software Primer example. But it is much more powerful than this simple illustration. It can compute our tax return (assuming we feed it our tax data). It can offer predictions about the behavior of the financial marketplace (assuming we feed it accurate numbers.)

But *it can only execute what the programmer has coded it to do*. It is not clairvoyant. Its "intelligence" is only as intelligent as the people who have written the programs. If Wall Street computations for predicting Wall Street behavior are flawed, so is the software. It is that simple, and at the same time, that complex.

Take a look at Software Example One. It consists of a few lines of code that are used to deal with a stuck accelerator pedal. It is hypothetical. It does not reflect Toyota's system and a car manufacturer would likely not use software to make the decisions used in these examples. But it is suitable as a model for our discussion and maybe not so fanciful after all.[3]

Software Example One.

Ongoing Operations Routine
IF (excessive speed is detected and brakes are being applied) GoTo Decelerate Routine. Otherwise, do nothing.
:
:
Decelerate Routine
Disengage throttle.

The two underlined statements are lines of non-executable code that allow the software to "branch" to specific parts of a program. Think of them as addresses to different lines of code. The italicized, bold text represents executable lines of software code.

[3] Some experts are predicting future cars and associated driving lanes will be controlled by hardware sensors and software logic. I am hoping these systems become realities *after* I become an unreality.

On the surface this software seems simple enough: Going too fast and you're trying to stop? I'll slow you down![4]

How much Software?

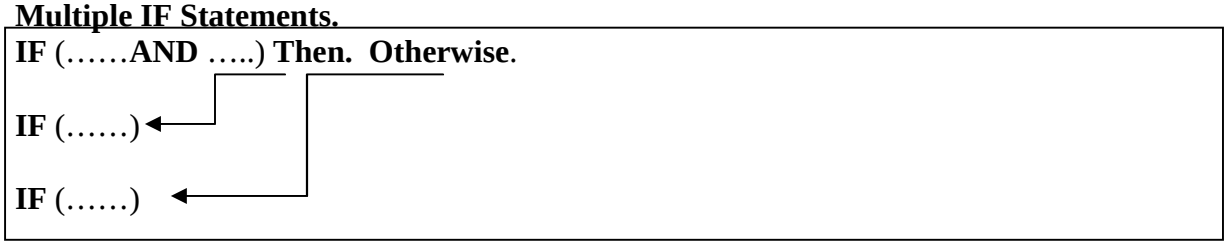
But if we imagine cars of the future in which it is predicted software will control all aspects of navigation, it is not so simple. What is excessive speed? It depends on the road, legal jurisdictions, and the weather. Not to mention acceleration onto an on-ramp; acceleration to pass a car, and so on. Therefore, the software and associated hardware detectors (temperature gauges, water detectors, etc.) must be designed and programmed to test a wide variety of variables with hundreds, perhaps thousands of these **IF** statements.

That's just a starter. What is the threshold for deciding if brakes are being applied? Because the brake pedal and its connected parts age, the software threshold of "being applied" changes over time. Even cars just off the factory floor may vary in how their brakes might be applied.

Anyway, even if these cars never appear on highways, existing software is written with a lot of **IF** statements, such as the example above. And here is good news and bad news: **IF** statements are the bedrocks of software power and software complexity. To understand why, let's try another programming example. This next box titled Multiple **IF** Statements uses arrows to show the flow of the logic of the software. That is, its sequential execution.

Many **IF** statements provide two or more alternative actions. In this example the **AND** allows the programmer to take into account another possibility. For example, the software might check if the Toyota is located on an Interstate with a 70 MPH limit **AND** operating under heavy rain conditions. If both conditions are met, the "branch" to the second **IF** statement might result in an alarm being sounded to the driver, or a forced deceleration that is taken over by the software.

Multiple IF Statements.



Not so fast! (so to speak). If the car is speeding along at 70.0001 MPH and the rain gauge shows excessive precipitation (and what is excessive?), there are other choices. Stop the car? Inflate the air bags? Call 911? You can make up your own scenarios as well as I. Whatever they may be, the possible actions become complicated.

Then there is the software **OR**: Has the Toyota been operating within a MPH limit, **OR** is it exceeding the limit, **OR** is the car speeding to an emergency room because the driver is having a heart attack---possibly from his fear of not knowing what his car is doing? *You're going too fast! Your software has put on the brake. Your heart attack can wait.*

[4] This example is just that, an example. But this simple logic may not have been in the Toyotas that have come into question. See a later footnote for more information: "Toyota learns the tyranny of software complexity."

The many decisions we make as we pass from Interstate 90, onto an off-ramp, onto a side street, or through a school zone are made through the marvelous semi-conscious synapse firings of our brains. These actions, when relegated to software, must be programmed with thousands of lines of code, often *millions* of line of code.

All well and good. Car brakes brake. Airplane flaps flap. Wall Street models model.

Not necessarily. Software is written by humans. Even with our superb minds, we make mistakes. These mistakes become part of the software that governs the flaps on airplanes, the accelerator mechanism in a car, or a model in a financial analysis program.

The Sins of Commission and Omission

Software, like humans, is susceptible to two sins: The *Sin of Commission* and the *Sin of Omission*. The Sin of Commission assumes an event---however unlikely---might occur. For example: **IF (the Dow Jones goes down 3,000 points within one hour) GoTo Rewrite-Will Routine**. This idea, as related to software is usually harmless, as it does not lead to problems, just too much code from an overly cautious programmer.

The Sin of Omission is the big problem, because like humans, it does not cover all bases. For example, consider the many variables that influence our decision to change lanes to pass a car, such as oncoming traffic; speed of our car and the car we may be passing; road conditions; weather conditions; visibility, and so on. For the sake of simplicity, let's assume our intuitions about this possible action are translated into code. *Every* possible condition had better be in the software. …*Opps, the programmer forgot to code an **IF** statement to check for oncoming traffic*.

Silly? For this scenario, yes, but many problems occur because software has not been programmed to cover all contingencies. We often read of "software failures" and their tragic consequences. Many of them are omissions of **IF** statements. We blame the "computer" for many of these ills. The fault lies with the person who wrote the code or the software analyst who made the decisions about what contingencies the software would and would not cover. Here is an example of the Sin of Omission as it relates to the Toyota problem:[5]

> The system level error that Toyota made is not letting a brake signal override a throttle signal. I designed speed control systems at Ford, and everything was dependent on having a tap on the brake cancel any speed control function. A throttle-by-wire car like Toyota makes is almost free to add speed control, you just have to have a button to tell the ECU (engine control module) to hold speed and a brake signal, and that is probably already sent to the module. So it was just software, a couple lines of typing, that means that once a car accelerates a brake input will send the throttle angle to zero. I have to assume that Toyota engineers talked themselves into thinking there are times when you want to hit the throttle and brake at the same time. Motorcycle racers do this to put torque loads on the frame so when they do let off the brake coming out of a corner, the bike is already "bent" by the chain loads and then handles more predictably. I can't think of a reason a car needs to have brake and throttle on at the same time, but

[5] "Toyota learns the tyranny of software complexity." Mar 1, 2010, 9:03AM | Permalink |Comments (30).

somebody must have dreamed it some sports-car-dork reason to not have the brake single cancel the throttle signal.

Branches and Decision Points

If the non-technical reader is still with me, you're over the hump. We can now expand the discussion on complexity in a less technical way. For starters, software statements often have many branches to other statements. These branches may change depending on the results of the **AND** and **OR** comparisons (and the **NOT**, which we are NOT going to discuss). The software might branch to a routine to compute tax refunds if we pay too much to the government or to a routine to compute what we owe Uncle Sam if we have paid too little.

Under the best of intentions, various combinations of branches can become very complex. A simple example is provided in the box titled Coupling of Code. The first **IF** statement results in a branch to a different part of the program, which may----depending on the conditions of the next **IF** statement---result in a branch back to the "original" code (the red arrow). By then, conditions might have changed, which would alter the flow of the execution of the software logic. By altering the program's executions, the movement of the wing flaps on the Boeing 787 might be affected…the plane in which we are seated as it tries to land during a windstorm.

Coupling of Code.

No problem…unless the software is suffering from the absence of a few lines of code (a Sin of Omission) and should have taken into account "unforeseen" sudden side winds or wind shear. The intuitive reactions of a pilot must be "intuited" by who knows how many lines of code.[6] *Stewardess, is it too late for another drink?*

But even in relatively simple software systems, the number of conditional statements with **AND**s and **OR**s, and the number of branching possibilities can number in the hundreds of thousands.

Let me repeat: Even in relatively simple software systems, the number of conditional statements with **AND**s and **OR**s, and the number of branching possibilities can number in the hundreds of thousands. …*Thanks for the gin and tonic stewardess.*

Look at the Coupling of Code box again. Multiply this simple logical scenario by thousands of permutations and you can begin to gain an idea of how complex software can be.

And by being complex, it becomes more susceptible to error. Not necessarily the errors of Sin of Commission or Sin of Omission, but also the inability to track all the possibilities of the paths of

[6] In fairness, we are told a pilot can override the software control. But to what extent? Some airplanes, even under supposedly direct pilot control, still have software acting as an intermediary between the pilot and the plane.

logic the software might take; especially as changes are made to existing software modules. Often, the addition of code to correct a problem creates yet another problem.

The Million-Line Baby

It is not unusual for a system to operate with millions of lines of software code. In sheer volume of sentences to read and understand, consider that a novel contains about 500,000 characters (100,000 words x 5 characters per word.) A million lines of code is equivalent to about 40 novels.[7] And novels don't have **IF** statements that branch the reader around different parts of the text. Their statements flow from Chapter One to the final chapter. Software does not behave in this simple sequential manner. Once the execution logic begins to jump around in the code, things can get complicated.

The Boeing 787 flies with the assistance of 7 million lines of code. Not to be outdone, a 2010 GM car, with its many microcomputers, cruises along with the assistance of 100 million lines of code.[8]

Recall those two lines of code from a Dell program that interfered with my new computer? Only two lines of code caused a major problem. Cars and airplanes operate with millions of lines of code. *Fasten your seatbelts folks.*

Parallel Processing

Like people, computers divide their work. Humans form teams to increase their efficiency and shorten the time to do the work. So do computers, which usually have more than one computational processor (Similar to a human work group having more than one human.). The box titled Parallel Processing shows a simple example of how the processors in computers divide their work.

Parallel Processing.

Line 1: A + B = C.
:
Line 5000: X + Y = Z.

Lines 1 and 5000 can be executed at the same time because the variables in the computations are not the same. Of course, if line 5000 were **C + Y = Z**, then line 5000 must be executed after line 1 in order to use the latest value of **C**.

Therefore, software in many computers will analyze this code and make decisions about how to "farm-out" the code in the program to different processors. In this straightforward example, the time to execute these two lines of code is cut by one-half. If a program contains a couple million lines of code, parallel processing can substantially reduce the processing time.

One of the biggest challenges in maintaining efficient and error free software is to make certain the ongoing changes to the code do not introduce errors. Why must changes be made to code? Because of initial design deficiencies, such as the Sin of Omission. Another reason is changing

[7] NASA paper, "Flight Software Complexity," NASA OCE Technical Excellence Program," nd.
[8] Ibid.

user requirements: *Maybe we should add some code that disengages the throttle when the brake is applied.* ☺

For purposes of illustration, code is inserted into line 5000 as **X + Y – A = W**. The previous parallel processing logical structure is no longer correct. Line 5000 cannot execute independently of line 1. Okay, this is easy to detect and circumvent. But if thousands of lines of new code must be added to millions of lines of existing software, including many including **IF**s and branches, the resulting programs might end up with errors.

In so many words, things can become very complex, for example, errors, as in two lines of code from millions of lines of code.

On our Humble PCs

How complex is the software operating in our PCs? Even though many of the software programs do not interwork with one another, it ia astonishingly complex. Apple's Macintosh operating system contains about 86 million lines of code.[9] I was unable to find a number for Windows 7, but my hunch is that it has comparable volumes of code in it.

What do we do?

Does sheer volume create complexity? Not necessarily, but yes, usually it does. Sheer volume means more and more functions are being delegated to software, resulting in more lines of code that can be incorrect.

I recently invoked McAfee to scan my new Dell PC files for viruses. I instructed McAfee to scan everything, not just "critical files." My machine is far from fully loaded. I still have several applications and games I intend to place on the PC. Still, with my new (semi) tabula rasa, the scan examined 146,470 files, 100,770 of which were tagged as critical. (My old Dell Vista PC had many more, but then it had assembled a lot of baggage over its life.)

Some of these files have thousands of lines of code. Thus, my PC easily contains millions of lines of codes and tens of millions of conditional statements and branches.

Now What?

Returning to my original stance, we cannot do anything about the software world we live in. It is part and parcel of our modern life. So, what do we do?

If you are migrating to a new computer, be prepared for some rocky roads. Unless, of course, you bring in the local Geek Squad to take care of your needs.

For the broader picture, we can expect more Toyota-type failures. I am amazed more failures do not crop up, not just in Toyota, but in other cars as well.[10]

[9] http://en.wikipedia.org/wiki/Source_lines_of_code.
[10] I remain a loyal customer to this company.

In the meantime, try to ignore that your flight from LA to NY is often under the control of software. Have a drink, and try to forget about software Sins of Omission, or for that matter, Murphy's Law: Anything that can go wrong, will.

For some good news. Today, I succeeded in re-loading onto my new machines an application that installed itself perfectly on the first attempt! It's my online Texas Hold'um from PokerStars.net. It's the *only* application I've loaded thus far that did not encounter problems.

I'm headed for an Internet gaming table. My interactions will be with real people, but overseen and managed by software.

Software Complexity and Software Failures (II)

April 10, 2010

A few days ago, I filed a report on the problems we face with the increasing complexity of software operating on our computers, as well as the software executing in other computer-based machines we use. As two examples, computers that control car brakes, and computers that control the flaps on airplane wings.

I hope the report was not too technical. I tried to keep it relatively free of geek speak. I thank you for your responses.

That said, I hope all readers will read the article and will not be discouraged by its title and content. (I received positive responses from computer neophytes.) If you have not done so, please read this report. If you use a computer, the facts I lay out in the report will be helpful to you. (Depressingly helpful but forewarned is forearmed.)

As with the first report, this report is not good news but contains information that you should know, if for no other reason than to realize many of your "so-called" mistakes while using your computer, smartphone, etc. are not your mistakes. Perhaps that thought alone will offer you some solace in dealing with our increasingly automated world.

As a brief summary of parts of the first report, I mentioned that the increasing complexity of software is leading to more errors in many of our daily transactions with computers; in some instances, with tragic consequences. I mentioned that poorly written software code was one problem.

This report explains another: A user's input (from you or me) might *change* the software code. *Code that might have otherwise been correct.*

User Interface

I use godaddy.com to provide my Web and blog services. The company also administers my Internet domain names (addresses), such as UylessBlack.com. I like the company's support. Its name has nothing to do with its professional attitude and performance.

Although I come from a software programming background, I now steer away from this activity because I wish to use my time in other pursuits, such as writing this report. Godaddy provides several well designed screens to assist me in keying in non-technical sentences about (a) what I want my blog to do and (b) what I want it to look like.

For example, let's assume I key in this sentence into a godaddy furnished window:

> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

Then, I use godaddy's color palate to make the sentence appear as:

©Uyless Black 2010

> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

I make this color change because I want the viewer to see the red lettering as a major subject.

(These simple entries can be made by most anyone, even my mother-in-law (no offence mom)).

Next, I drag my mouse across some of the text to highlight it. The bold "modern art" words symbolize my highlight:

> The Confessions of a Modern Art Luddite explores **modern art**, especially abstract expressionism.

I then go to another godaddy window and paste "modern art" into a slot. I am asked to associate this paste with a file. The file could be a PDF document, a photo, a video, etc. My clicking on this request associates my paste of "modern art" to a file I had downloaded earlier.[1]

The software changes my sentence to appear as:

> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

Now, "modern art" is a link that associates its name to my file. Thereafter, when a viewer clicks on modern art, the file is automatically sent to the user's computer.

It is very simple. All I do is enter English text and follow godaddy's commands to create a link to one of my files (reports).

Resulting Code for Powerful Results

The simple sentence I entered onto the screen is not software code. Godaddy translates my sentence into code that is executable on a computer. The code created is called HTML some other language.[2] It is gibberish to all people except programmers, so I will not include it in the main body of this report. (See Appendix.)

The important point is that even nonprogrammers can use godaddy's interface to create fancy and powerful Web and blog sites---by just keying in English-like sentences.

Unintended Changes

A few days ago, I was entering additional text for my blog. I was also downloading reports to set them up to be linked to words in my text. These changes added more HTML code.

[1] The download is also easy. The software allows me to tell it I want to download a file from my PC to the godaddy server. It allows me to browse my PC and click on the file I wish to be downloaded. For this example, I click the appropriate file. Then, I simply fill-in some godaddy fields that associate the term "modern art" with my file.

[2] Hypertext Markup Language.

©Uyless Black 2010

However, my changes also resulted in altering some of godaddy's code! I use an exclamation point with good reason. I was *not* changing the code. Yet, my entering of text had the result of changing some of the HTML code that had been created previously.

Vendor's Response

The alterations to the HTML code were not serious, but they were a nuisance. For example, below is what happened to the HTML code that reflects the statement I discussed above. First, here is the correct text:

> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

My ***non-programming*** changes resulted in the red text and the link text changing to black text:

> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

The link "modern art" still worked correctly, but the blog was less user-friendly.

Keep in mind that I made no changes to code. I only entered text and used godaddy's interfaces to create links to my reports.

Godaddy's technicians thought I had gotten into their HTML code and changed the color parameters of many text variables. I had not touched the code, which caused great confusion on their part. *They had no explanation about how the code could have been changed by my non-coding changes.*

Unintended Consequences

A user's action that results in changing code is downright scary. It's akin to putting our foot on a brake pedal which results in altering the code that is controlling the brake. The code may have been correct in the first place. Our action changes the code to be incorrect; say, in not applying the brakes.

Is that idea bothersome to you? Let's try another. It's akin to an airline pilot moving the control wheel which results---not in altering the flaps on the wing, but in changing the code that controls the flaps on the wing.

It's bad enough that software is complex unto itself. It's even worse if users' non-programming actions result in the unintended alteration of the code.

My example is a simple one. We have more serious instances of security breaches in which files are altered, or code is changed or destroyed. Mine was not a security breach. Mine was an unintended change.

Sleep well, the software governing your welfare may be as pliant as your pillow.

©Uyless Black 2010

Appendix: Geek Speak

Some of my readers who are software knowledgeable might offer that I am creating a straw dog. That:

- A well designed system would not have allowed me to intrude into the vital code that turns the wheels of the application.

- Even if the intrusion occurred, my changes would have had to go through a compilation process…which would have rejected my spurious alterations.

- Even if my changes took, security software would have then blocked them from executing.

I disagree…obviously because I created the arguments in the first place. But more: First, I did not construct a hypothetical situation. Second, it is well-established (but not well-published) that systems have been breached by hackers, resulting in attacks on the system's software.

My point is simple: Software has becomw very complex. It may be incorrect when it is first written. It is one thing for it to be incorrect to begin with. It is another that we, as unwary users, change it to be incorrect---after it has been validated to be correct.

For the software experts, below is an example of some text I entered into my vendor's screens:

> For additional pearls of knowledge, click on Slims Wisdoms.
>
> The Confessions of a Modern Art Luddite explores modern art, especially abstract expressionism.

This is the code my vendor created from my text entries:

```
181307/01_Slims_Wisdoms_(I).pdf">Slims Wisdoms</A>.
</SPAN></SPAN></SPAN></SPAN> </FONT></SPAN>  <BR><
BR><SPAN style="COLOR: #bf335a"><FONT color=#000000><SPAN
style="COLOR: #0a0a0a"><SPAN style="COLOR: #bf335a"><SPAN style="COLOR:
#0a0a0a">The <SPAN style="COLOR: #ff0000"><FONT
color=#bf335a>Confessions of a Modern Art
Luddite </FONT></SPAN></SPAN></SPAN></SPAN> explores
</FONT><A href="<BlogInfo:URL />/files/7/0/3/1/8/192009-
181307/Confessions_of_a_Modern_Art_Luddite.pdf"><FONT color=#000000>modern
art</FONT></A><FONT color=#000000>, especially abstract expressionism.<
```

My non-code changes had the result of changing some of this code.