There has been talk of technology and bringing Carroll up to the twenty-first century. Many people do not understand technology and some that try do not understand how to deploy it properly. The end result is a waste of money, effort and because of the failure they then blame the technology itself and proceed to use more Luddite methods instead. Technology itself is not a ‘thing’, nothing exists that you can point at and say ‘See that bit there, that is Technology’. Like all other ‘-ology’ words, ‘Technology’ is the study of skill or productive work. If your efforts to deploy a system isn’t productive it isn’t technology.

To begin with, we need to teach technology in our schools. When I was in High School and Vo-Tech/CCCTC, I took classes in Electronics. Other trades and sciences were taught as well; like carpentry or civil engineering. However over time some technology courses were replaced. The Electronics course that taught circuitry, components and design was switched out for 3D printing and manufacturing processes. So the knowledge of what made technology work was replaced with how to put tab A into slot B; basic mechanics and assembly. This does not teach technology, a student would not understand how to make a device ‘do’ something and only knows the cursory value of how it was put together.

Combine this with the effort to teach students software by proprietary companies using ‘toy’ software, in some cases literally, they then learn wrong methods for developing software that is installed in those same devices they don’t know how the functions work. So you end up with no skills to produce quality work. This failure is masqueraded by the fact that the ‘tool de jour’ is changed every six months. So you lose your skill set for another and never properly learn either.

We need to stop this fashion trend in the name of STEM and focus on actual skill-sets and tools. Technology is boring unless you have an interest in it. STEM turns technology and science into a game that is never used in real life.

My actual work experience has shown there are two main branches used by actual developers. One is Windows-based and uses Virtual Basic/Mono for software development. The other is sometimes Windows-based other times Unix-based (Mac,Linux or BSD) and uses C (ANSI or ISO) or Java for software development. Hardware-wise, only Electronics, Mechanical or Electrical Engineering experience is used to develop hardware. 3D printing is just a prototyping system if it is ever used at all; mostly assembly is done by hand. The Assembly language is used in firmware and micro-controllers (due to limit resources).

No one uses STEM, no employer asks about STEM, no one uses Perl or Python in a production system and Java is usually not used in production processes but is sometimes used in the final products.

The other problem is the confusion between who is developing technology and who is using it. Throughout the so-called Open Source community, this line is blurred; not because a person must be both but rather they expect it of you. As a result we have an industry called ‘IT’ that means absolutely nothing and has become an umbrella term for computer technology as a whole (and has started to absorb some other industries as well).

Was this has caused to occur is a niche industry of software developed for one purpose, not as a focused tool of quality work but rather as a philosophical agenda and if you don’t agree with such philosophies then there will be no cooperation in developing the tool for your work purposes.

Within corporate environments, software is seen as property and not a tool. This limits any adjustment to make a tool fit for a purpose. You are instead expected to know before hand which tool is needed for which job. Unlike actual hardware where the design of tools has been established for centuries, thousands of years in some cases; software has no established design or purpose. There are some common utilities for basic operation on data (move, copy, delete, etc.) but on the whole there is no standard and no cooperation on creating one. It’s the old engineering joke; ‘The good thing about Standards is there are so many to choose from’.

The problem rises mainly from that these tools are seen as proprietary property of a corporation or the issue of ‘Information Technology’ where there are an infinite number of ways to accomplish a limited number of tasks. Remember that technology is the study of skills or work; there is no work produced by information, so ‘IT’ is an oxymoron.

Open Source was attempt to get an end-work-around the issue of proprietary ownership using existing copyright laws. This is a problem as it still assumes the design of software is property. Software is like Songs, they are made of functions as music is made of notes. No one can own a note but they can own the collection of notes that make their music. This does not exist in software and has been the subject of recent lawsuits between Google and Oracle over the Java API (effectively Google wrote their own API replacement and Oracle didn’t like that but the courts said it was okay since they didn’t copy or duplicate Oracle’s work). Essentially Google wrote their own arrangement (how the song is played on an instrument).

The libraries (a collection of functions) and fundamental software to build programs (compilers and linkers) need to be open and public. Just as you can buy any tool and parts to build your own devices, structures or machines. You need to be able to obtain the libraries and compilers to build your own software. Both designing physical structures (carpentry) and writing computer programs (coding) are Trades and blue-collar industries. Just take a look at how some companies treat and abuse their programmers, it’s clearly a manufacturing industry.

Finally we need to address the platform where all these programs are executed. Like my music analogy, all programs need an instrument. There are two main types of platforms; micro-controllers and computers. The micro-controller is a small chip usually with Assembly language software that executes the program on startup (when power is applied). Your cellphone, Raspberry Pi and most video game consoles are micro-controller based (so is the radio in your car or mp3 player). You know if you have a micro-controller device when it is described as needing a 'firmware' update. That term is becoming more generic but it means the program stored in the micro-controller's memory.

The second type of platform is an integrated system like your personal computer. It contains many more functions than a micro-controller and may include several micro-controllers as part of the integrated system. Some can argue that the Raspberry Pi is a computer because it contains additional components but I don't as it is still built from a micro-controller that requires firmware to operate and load the OS. Some gaming console in the current generation could be described as computers (but they are proprietary).

That is the main distinction, the Operating System. It is a set of programs and libraries upon which all other software on your computer functions. The difference between firmware and the OS is that firmware mainly drives the hardware of the device, while your OS drives the software on your computer. This essay is being written using a word processing suite that draws libraries functions form the OS and its own libraries.

The point I am trying to make is that every device has its own firmware or OS and there is no standard or cooperation. Your Android phone uses different formats from iOS from BSD or Windows. Then you have to get these devices communicating with tablets, smartphones, 3D printers and other proprietary devices that do not reveal sufficient technical data to understand how they work.

This is where IT became dominate through the use of the internet (or local networks) to have these devices speak to one another. Now the current problem; the reliance on IT for inter-device communication to produce work. Because we didn't develop standards or public protocols, we are reliant on a cottage industry pretending to be mainstream for basic functions and processes. Every conversation about technology now revolves around the Internet and what to do when service is lost. This also leads to conversations about access to databases and who owns or has authority to view or manipulate data.

Due to this reliance on networks, if we don't establish a county-wide fiber-optic or other broadband system, we can't develop any more advanced systems or technology. Not because we need the internet for advancement but rather our tool-set is built that way.

There are at least three main areas we need to focus on to bring Carroll into the 21st Century era, a world we will be joining far too late.

First, we need to build an infrastructure of at least two parts; a fiber-optic network of at least 1.5 Gigabits per sec and a Wi-Fi system (at least 1 Megabit per sec) to support it, both wireless connections locally and to link in more rural areas. This will be a dark network (no active light) and we will sell access to providers and use the fees for maintenance of the network. Service should be available for around $35 dollars a month. This would be comparable with service found in other countries that have far more network availability.

Second, we need to bring our tech curriculum in schools up to date. We need to teach computer science and programming in high schools using existing tools found in the industry. Instead of toy languages (like Scat or Python) we need to focus on common and best practices using C, Java or VB/Mono depending on target system. A focus on Unix would be better than Windows due to security flaws and proprietary ownership of tools and code. So that means using ISO C for programs, Perl (or shell scripts) for interpreted code and Java for cross-platform programs (despite its problems it is still the best choice for this purpose).

There is a set of books called the “Write Great Code” Series that does an excellent job teaching not just how a computer program works and how computer data is manipulated but also teaches best practices for development cycles as well. It uses tried and true methods found in Engineering (not IT) by an author that written several compilers, taught software in university and has literally written the book (several times) on Assembly Language and Compiler development.

Third, I’d like to build up an actual capacity within the Dept of Public Works (or more appropriate agency) to develop software and hardware for county needs. We cannot rely on Corporate America to sell us solutions to problems that their products are designed for rather than the problems we actually have. I do not want this capacity performed by the Office of Technology Services whose mandate is to deploy technology and teach its use to staff. That is a different mindset than the actual development of the technology. Also the OoTS is focused on IT and not Engineering.

Where appropriate I would like this software to be public property available to Carroll Residents for their own use. This would open up opportunities and allow people to being able to use technology for their own purposes much as how existing hardware tools (hand or power tools) serve communities. Business would have to pay for support and access to these tools for maintenance and they have corporate products that they could buy, the average resident does not have such access.

The above three points is the minimum I’d start with and develop capabilities within our community, students and local businesses from there. I would not focus on so-called ‘smart’ solutions brought on by ‘IT’ for problems that do not exist.