

Fall/Winter 2010

By Warren Loomis,
President & CEO Versaterm



It's funny how technology goes. If you are around long enough, you begin to notice how the industry continually re-invents itself over and over with the latest trend billed as the "savior" to all of your woes. What's interesting is that the latest trend is usually built on methods that were used in a generation that was once or twice removed from the present generation (i.e. grandfather or great-grandfather generation). Personally, I have been around for a generation or two and have seen a couple of full circles (actually, a technology generation is between 5-10 years so I've witnessed a couple of them). What is truly amazing is the hype surrounding the latest method and, more often than not, its inability to recognize what really worked well in the previous generations and what didn't work so well. Hence why the latest fad tends to be praised as the be-all and end-all fixing all of the woes introduced by the previous technology methods. Much like generations of people, the "teenage" technology methods fail to learn across generations but simply looks at the "parent" technology as obviously wrong.

This article is not about specific technology (e.g. HTML, JAVA, .NET, etc.) but will focus on the "technology methods". The technology methods are described by how to deliver and manage the applications (and services) to the user. The tools or languages used are often the next best thing to deliver those methods but should not be confused with what the industry is trying to accomplish: develop and deliver manageable, cost-effective applications with rich functionality that will solve a problem. For years, the industry has been trying to reverse the "80/20% rule"; traditionally companies and institutions spent 80% of their resources keeping existing applications running leaving only 20% for developing needed new applications.

***The best public safety software...
but don't just listen to us, listen to our clients - we do!***



VERSATERM

2300 Carling Avenue Ottawa, Ontario, Canada K2B 7G1

PHONE: 613-820-0311 FAX: 613-596-5884

EMAIL: info@versaterm.com www.versaterm.com



When I graduated from college (the late '80s), the industry was still primarily delivering "green screen" applications to the user. These systems were mainframe or minicomputer based and delivered usable application functionality to the user while being reasonably simple to maintain (all of the business logic was contained on the server so was easy to repair or upgrade). As many may remember, although those mainframe-based systems were easy to manage, the applications were not always the most user-friendly or intuitive as they needed to rely on the keyboard and keyboard shortcuts to navigate the application. But, as MS DOS was the operating system on the PCs at the time, the green screen applications fit right in. Although Apple's Mac and X-Windows was available, neither had the penetration of the IBM PC (or clone) running DOS with WordPerfect (with the keyboard templates) and Lotus (with endless key-combinations) being the core "office" products in widespread use at the time.



Phillip Seymour,
VersaCAD 1 Prototype,
circa 1980

By the early '90s, IBM, in partnership with Microsoft had been attempting to build a new PC core operating system called OS2, building off of the successful DOS partnership they had enjoyed over the past decade or so. Eventually, Microsoft felt they could do it better alone and spawned the first commercially usable Windows product in Windows 3. The more intuitive user interface became the de-facto standard and developers began to scramble to develop applications that would be seen as "modern" in the Windows environment. To take advantage of this new paradigm, the application development had to "leave" the backend mainframe-type computer and reside on the PC itself. TCP/IP became the standard network protocol and databases quickly supported new standards in client connection methods, serving PC-based applications from the central data base.



*The best public safety software...
but don't just listen to us, listen to our clients - we do!*



By the mid-nineties, users were rejecting any applications that did not fit the Windows mold via a powerful graphical user interface. This generation of GUI applications was known as “fat” clients as the business logic resided entirely on the PC and connected to a central database server. The applications were “rich” in functionality and could leverage the Windows operating system and all of the features available (e.g. document scanning, etc.) – something unheard of with “green screen” applications as these were agnostic when it came to the client (e.g. PC or dumb tube) running the application. The industry “ran” to this new paradigm like lemmings but what the industry failed to predict, and was slow to appreciate, was that by placing all of these applications on the PC, it became costly to support as technicians often had to visit each PC to perform upgrades and patches. Whereas in the mainframe era an application could be patched or upgraded once, it now had to be patched/upgraded on 1 to n PCs. At the same time, the networks were neither robust nor fast and the constant movement of data between the PC and the database server for processing resulted in poor network performance. To counter the network saturation of data, databases became programmable so a portion of the load could be run directly on the server (i.e. the business logic started to split between the PC and the database server). This addressed some of the performance problems but did not fix the manageability issues or address the fact that the prevalent Windows version was neither robust enough nor, at the time, did many of the PCs have enough horsepower to effectively run many of the new feature-rich applications. The result (for most) was a very high support labor and PC replacement cost.

Again, the industry responded and found a new technology that would be better than the previous. By the late '90s, the industry recognized that the “fat” client approach was *all wrong* and the “reaction” was to adopt the latest trend in Internet technology - everything should be HTML-based and delivered through a web browser which effectively reverted back to the server-based technology using the Windows PC as a “dumb tube”. This was essentially “green screen” architecture but with a much prettier and more intuitive user interface (i.e. based on the success of the “grandfather” technology methods). The advantage was the application could be upgraded/patched once at the server and the user received it immediately – technicians didn’t need to touch each PC.

The problem was that HTML and the web browser was not designed, at the time, for rich applications and the “page mode” (i.e. full page refreshes) consumed network resources and was slow to use (might be alright in casual use or surfing but accomplishing significant data entry was really quite painful). Further, the web browser was meant to be client agnostic and was not supposed to know anything about the device and operating system it was running on. Thus integration with the Windows operating system was lost! Javascript was a partial answer but it was never designed to handle big applications – the interpretive language was not fast enough.

About that time, the industry came up with yet another answer. Leveraging web technology, a new language called Java promised to deliver the rich application to the user, was housed on the central server for manageability and was client device/operating system agnostic. By the late '90s, Java was the savior as its architecture fixed all of the past woes.

We won’t get into the merits of Java but the real problem was that it was never designed for every type of application – it was originally designed as an appliance language where you build once and deploy on many different devices including PCs, refrigerators, etc. Java was truly device agnostic and was never to know anything about the host operating system or device – hence it was difficult to take advantage of the features in Windows. Finally, as it used a JVM (Java Virtual Machine) to execute code, it was very slow if, for no other reason, the JVM was not originally designed to meet all of the expectations the industry placed on Java. Remember Corel’s WordPerfect on Java? Probably not as Corel spent millions to create a Java version of WordPerfect but never delivered a commercial version as performance was abysmal.

***The best public safety software...
but don't just listen to us, listen to our clients - we do!***

3



VERSATERM

2300 Carling Avenue Ottawa, Ontario, Canada K2B 7G1

PHONE: 613-820-0311 FAX: 613-596-5884

EMAIL: info@versaterm.com www.versaterm.com



With Java failing, a new crop of languages emerged including the .Net architecture from Microsoft which, interestingly enough, is very similar to the Java architecture (.Net is essentially a JVM but closer to the Windows operating system). As Java was also extended to be aware of the client device (PC and Windows OS), the .Net application architecture provided the best of both worlds where the business logic could be essentially housed on the server with a rich client application architecture being aware of the Windows OS but without the headaches of the “fat” client manageability issues. The downside of .Net is that it is not portable across operating systems which, given Microsoft PC market share, is not the end of the world.

But the world is ever changing and user expectations are changing too. The cellular phone has morphed into the “smart phone” and with faster and faster cellular and WiFi networks available, the population is expecting more and more functionality “on their hips”.

The early smart phones came with a few rich front-end applications (such as email clients, word processing, etc.) but expected the user to use the supplied mobile web browser for accessing applications remotely. Have you ever tried to use the mobile web browser? It can be quite painful and reminds us of the ‘90s when the industry expected that the web browser and HTML was the answer. The Java and .Net architectures simply aren’t well suited for the smart phones as they are designed for more powerful devices and the form factor provided by the PC.

But along comes Apple, with the iPhone, and it seems they have come up with the next generation of technology methods. Recognizing the user really can’t use the smart phone’s web browser to effectively and efficiently run applications, the App method provides the best of both worlds, where the data and logic remains on the server while the user has an efficient client application on the front-end. For me, the light came on when I saw my bank advertise their App to access bank accounts and pay bills. Having tried to do these functions through my smart phone web browser (and giving up), the App delivers. The underlying architecture may be similar to the .Net and Java but it is designed from the ground-up for the next generation of user interface and devices. Apple didn’t try to port their Mac OS X to the iPhone and use the Mac applications but realized that a new device paradigm would require a new technology method.

The app architecture is different from traditional development architectures as the App is typically single purpose (not a full system) and is designed to be small, easy to download/configure, quick loading and as they are single purpose, very quick to develop and test. They are not device agnostic but leverage the client operating system and device features. Given the plethora of Apps now available for the iPhone (200,000 and growing), one could conclude that they are solving a real problem for mobile users.

If you want an app to tell
you what app to get,
there’s an app for that.

Apple has subsequently extended the “app” method to a larger form factor in the iPad and, as the operating system evolves, that could be the technology method of the future.

*The best public safety software...
but don't just listen to us, listen to our clients - we do!*



Other smart phone operating systems and devices have since followed and offer Apps for their devices. In time, PCs may have to adopt and support the App method as it is proving to be an efficient and effective method in delivering and managing applications. If the method can become a cross-platform bridge between the smart phone and the PC, then applications can be extended from the PC (in the office) to the mobile user (smart phone) in an effective and efficient manner - blurring the lines between the fixed position desktop and the mobile user. The difference in the approach is that the App technology method will build from the smart phone out to the PC and not try to fit the PC method onto the smart phone – a novel approach that could be a game changer.

Is the technology method that different or is it built upon legacy technology? The majority of this article actually demonstrates how there is nothing that new but each “new” technology method is built upon a previous generation - just not the parent generation. Although the Apple App architecture is not necessarily new, as explained earlier, it is very different. The App technology method is built upon the idea that the user is (primarily) mobile and not sitting at a desk. The upcoming Apple operating system codenamed Lion will apparently support App technology – this might be a sign of things to come.

Is the App architecture the answer to all of the system development and management problems? I would hope that after all of the years of blindly running to the latest technologies as “saviors” or “silver bullets”, the industry has learned and would answer “no” to that question. The App architecture fixes a real problem for mobile users and may even extend into the desktop environment as the desktop and mobile lines begin to blur. However, the industry should not expect it to be the answer to every problem and if they do, then we are destined to repeat past failures.



Warren Loomis

*The best public safety software...
but don't just listen to us, listen to our clients - we do!*

