[](https://hackernoon.com/%40terrycrowley?source=post_header_lockup)

[Terry Crowley](https://hackernoon.com/%40terrycrowley?source=post_header_lockup)Follow

Dec 12

**Why Windows on ARM?**

The [story line](https://www.theverge.com/2017/12/5/16737288/microsoft-windows-10-qualcomm-arm-laptops-launch) for Microsoft’s recent release of Windows 10 on ARM is that they’ve learned from the [Windows RT disaster](https://www.theverge.com/2013/3/21/4004540/intel-vs-microsoft-how-atom-processors-could-kill-windows-rt). Those idiots, what were they thinking back then?! As is usually the case, the real story is more nuanced than “they were idiots” — and I think more interesting. It also has consequences for the new release and explains some of the strategy behind these recent releases.

As with my [Vista post](https://hackernoon.com/what-really-happened-with-vista-4ca7ffb5a1a), I’ll caveat this with the statement that I was not in Windows leadership meetings as these decisions were made, but I did have long and regular meetings with Windows leaders during this period as we worked out plans for Office on ARM devices during the Windows 8 — Office 2013 development cycles.

Let’s roll things back a bit to get some context. The threads here involve early tablet efforts, the iPhone, the iPad, Chromebook and the challenges in the overall Windows ecosystem.

Microsoft had tablet ambitions and efforts for quite some time. While early prototypes and investigations used different processors, ultimately Microsoft developed a tablet reference design in 2002 that led to a variety of Windows tablet computers built on Intel x86 chips. The tablet computers added features for stylus input and inking as well as convertible form factors that allowed keyboards to be detached or folded under the display. These were full Windows PCs with all the benefits — and disadvantages — of regular PCs. This meant they were relatively heavy with unexceptional battery life but could run the full spectrum of Windows applications.

While the stylus and ink input opened up new scenarios for these devices, in practice the constraints of weight and battery life meant that there were not broad new markets for these devices. In addition, only a few applications were written to take advantage of the inking capabilities. The lack of more applications for this form factor was more an effect of the limited success of these devices than the cause.

The theme through the 2000s were gradual improvements in laptop weight and battery life but no breakthroughs that would open up major new scenarios and markets. Intel released lower power processors which were integrated into “netbooks”. These were inexpensive but clearly simply inferior PCs. Windows had gone down a path with Vista of radically increasing system requirements over Windows XP rather than trying to reduce resource requirements and so was poorly positioned to target these lower-powered devices. In fact, the combination of low-powered netbooks and power-hungry Vista ended up forcing Microsoft to allow PC manufacturers to install a non-current version of Windows, Windows XP, on these devices.

Perhaps most critically to this story, Windows was experiencing a high degree of “ecosystem rot” during this period, especially in the consumer space. A typical consumer PC would come loaded with “crapware” that killed the initial user experience and performance of the PC out of the box. This shoot-yourself-in-the-foot behavior by Windows OEMs was directly related to the economics of the PC business where Microsoft and Intel sucked most profitability out of the overall stack. Even getting pennies from software makers to pre-install trial software could make a difference in the overall profitability of these devices.

Downloading new software for a PC was a harrowing experience, with even reputable software often coming with additional add-on software like browser toolbars that cluttered the experience and dragged down the overall performance and reliability of the machine. Any reasonably technical person during this time period was familiar with spending hours during the holidays trying to disentangle and cleanup a parent’s or other relative’s PC.

The challenges here were directly related to the wide-open nature of the PC platform that had played an important role in its overall success. The software installation process could effectively do anything to the machine and a running application also had almost unlimited access to machine resources and user data. In fact, some of the most common software downloads for anti-virus and anti-malware were inherently some of the most invasive to the normal operation of these devices. The uninstall process depended on each vendor getting the logic right, which overall resulted in an unreliable experience. This meant that often the best approach for cleaning up a PC after an unwise decision to try some new software was just completely paving the PC and starting from scratch.

The wide-open nature of the PC had served it well in the early days but was collapsing under the weight of bad actors, unaligned incentives and poor enforcement mechanisms throughout the ecosystem. All these challenges were a very visible part of popular culture with Apple’s series of clever and successful Mac vs. PC ads.

Along side all these problems in the native PC application space, web applications were growing in both capability and performance. Web applications did not suffer from any of the acquisition and installation (or uninstall) challenges of native PC applications. The cloud architecture inherent in web applications also enabled exciting new scenarios. Google was especially aggressive here. Gmail was released in 2004 and revolutionized the consumer email space. A bigger impact on developer mindshare was the release of Google Maps in 2005. Here was a highly useful, graphically rich, responsive application built on top of the browser and remote cloud services. It served as a striking example of the kinds of user experiences that were possible on top of the browser, especially when compared to both other web mapping services like MapQuest and even native PC-based applications like Microsoft’s own Streets and Trips.

The mobile device space had been growing in capability but was revolutionized by the release of the iPhone in 2007. The killer app for that first phone was the inclusion of a real web browser. At the time, other phones only had very limited browsers that understood limited dialects of HTML. A real browser with responsive zooming and panning suddenly made the web available in a mobile form factor. The iPhone also synergistically aided the takeoff of gmail. While business mobile phone users were familiar with accessing cloud-based email through Microsoft Exchange or Blackberry, the capability to easily access your email from phone and desktop had not been widely available in the consumer space. Gmail and iPhone support of the IMAP email protocol enabled consumer access to this capability for the first time. Microsoft’s own Hotmail would not support easy iPhone connectivity for another two years.

The iPhone, like most phones, was based on [ARM processors](https://en.wikipedia.org/wiki/ARM_architecture). We were set up for a classic disruptive battle between low power, low performance, low margin, but high volume ARM processors and high power, high performance, high margin, lower volume Intel x86 processors. More than anything else, it would be Intel’s love of those high margins that would limit their efforts and progress in the low power space. The iPhone demonstrated that even with the lower performance of ARM processors, it was possible to create a high function delightful device.

The opening of the Apple App Store a year after the release of the iPhone was a revelation. By curating submissions to the App Store, Apple could enforce quality and behavior guarantees for applications — in stark contrast to the jungle of the Windows ecosystem. Apple fully owned the application acquisition, install, update and uninstall process so could guarantee its safety. Applications were sandboxed to guarantee isolation except for carefully controlled exceptions that could be managed through user settings.

Especially interesting was how Apple implemented OS capabilities to ensure the performance, responsiveness and battery life of these devices. In early versions of iOS, only the foreground application was allowed to run, preventing background activity from eating battery, network or impacting foreground responsiveness. Unresponsive applications would be aggressively terminated. Instead of having every application pinging the network for updates or notifications, Apple implemented a shared notification service and a very limited badge count icon overlay mechanism (and later more sophisticated notification capabilities).

It was the combination of the hardware and these software and policy approaches that allowed Apple to construct an ecosystem where the set of applications could explode while still maintaining control over the overall user experience. As the hardware and software controls improved, Apple would over time relax some of these constraints (e.g. when adding multitasking and constrained background processing).

Windows 7 was released in summer of 2009, two years after the iPhone came out. Sinofsky had led a major reorganization of a dysfunctional Windows organization over the previous 3 years and delivered a release that dug Windows out of the hole Vista had left it in. Windows 7 was a highly successful release and unlocked a wave of enterprise deployments that had stalled with the Vista release. Office 2010 also rode that wave of updates.

What Windows 7 had not addressed were the issues with ecosystem rot and the challenges posed by mobile devices built on ARM processors. These were growing in capability while maintaining a differentiated power envelope that would enable smaller, lighter devices than ones built on Intel’s x86.

While Microsoft had been delivering ARM-based Windows Mobile devices since the late 90’s and had successful keyboard-based phone devices with significant market share, it took till late in 2010 before there was any response to the iPhone with Windows Phone 7 devices. These were competitive devices from a functional point of view but were not aligned at all with the desktop application or API strategy.

Just a few months after Windows 7 shipped, Apple announced the iPad in January 2010, shipping in April. I remember going to the Bellevue Apple store to try one of these early devices. My first reaction was “Fuck.” This was not some awkward compromise. This was a light, responsive, delightful device. Instead of moving down from a desktop form factor, Apple had moved up from the iPhone form factor. The phone’s laser focus on responsiveness and battery life carried through to this new machine and enabled a breakthrough in weight and overall size for a tablet device. By moving up and remaining compatible with the iPhone (albeit in a somewhat clunky 2X mode for non-modified apps), Apple could leverage and build on their growing app catalog. In building up from the phone, they had hardware, an OS and an application ecosystem that were all aligned to deliver a great experience on these mobile devices. The functionality, weight and battery life finally delivered the suite of features to define a new category.

Windows 8 was already in planning before Windows 7 shipped and before the iPad was revealed, so there is no clear “Apple did this, then Microsoft responded” timeline. Microsoft had been frustrated with lack of innovation in PC form factors for a while (of course this was really a direct consequence of the economics of the PC business) and had considered building their own devices. As Windows 8 started to jell around a big investment in touch-based interfaces, the decision was made to build both an Intel-based tablet and an ARM-based tablet that could demonstrate integrated innovation in the hardware and software stack.

Windows would also take back control of the Windows API from the C# managed code diversion of Vista days and invest in a modern native Windows Runtime API. This WinRT API would expose a sandboxed application environment and include a full app store for acquisition, monetization, install and updates like Apple’s app store. This API would be designed for building touch-based, power-savvy applications.

The Intel-based tablet would be a full touch-based PC — able to run all legacy (Win32) applications as well as the new WinRT apps.

The target for the ARM-based tablet was a device that was fully competitive in weight and battery life with the iPad. The challenge was that the overall experience of the iPad was achieved through a combination of the hardware, the OS *and the applications*. Win32 applications did not have the constraints and guarantees of responsiveness and power behavior that were required to deliver a competitive experience. Even if the Windows ARM device could run all those legacy Win32 applications, they would drag down the actual experience of using the device. Additionally, they would expose this new device to all the rot of the existing ecosystem.

So the key dilemma was framed as being between these two choices:

1. Do you differentiate by leveraging that enormous base of Win32 applications but at the huge risk of creating a device that does not actually deliver a competitive experience — and never will?
2. Or do you build a competitive device but abandon that Win32 moat?

Creating an obviously inferior device was a non-starter. The move to touch-based interfaces seemed like an instance of those rare once-in-a-generation inflection points that Microsoft *had* to leverage to move users and developers to the new WinRT APIs and experiences built on them. This was a way out of the rotten application ecosystem that was both Window’s biggest asset and its biggest liability.

The final decision to abandon that Win32 moat on these ARM devices included the decision to port the Office desktop apps as the only Win32 apps allowed on these devices. (Although competitive browsers were also allowed access for antitrust reasons.) The Office applications would serve as unique differentiators for these tablets as productivity devices. Microsoft also held off on shipping Office on competitive tablets to maintain this differentiation.

For Office, porting to ARM was technically straightforward — much simpler in terms of code changes compared to moving to 64-bit Windows which we had done in the previous release. Most of the coding investment involved being better power citizens on these devices. This mostly required externally transparent changes that allowed the processor to sleep for longer intervals without waking on some timer. There were other changes to be more aware of when we were using metered data networking as well as to support downloading only partial datasets (e.g. only a subset of your email). We also did work to make sure we were leveraging hardware support for video and audio codecs. Almost all these changes applied to both ARM and x86 devices (tablets and laptops), so there was plenty of motivation for these efforts beyond our ARM clients.

The other big change was a major architectural change to better support touch-based interfaces in the main Office applications (especially Word, Excel, PowerPoint and OneNote). That is probably worth a separate post on its own but helped set us up for a consistent architecture across all our touch-based platforms as well.

For the folks that suffered through it, there was an immense engineering challenge involved with testing and validating on these devices as the hardware, OS, tools and applications were all going through radical development concurrently. I was a growing fan of the “continuous stability and integration” approach that was a hallmark of the agile movement and that we were in the process of implementing in Office. I tried (with only limited success) to get the tools and OS on board with the approach. They were still fans of expensive staged updates and we frequently paid the penalty with the drawn out costs of integrating new updates and trying to coordinate linked changes across the Windows, Office and Developer divisions.

For all the abuse that Windows 8 would later take, its worth just summarizing how ambitious it really was.

* API. It would implement a completely new API, WinRT, with the OS infrastructure for a new sandboxed process and application model.
* Shell. It delivered a new touch-based start screen and shell especially designed for tablet devices.
* ARM. It delivered a complete implementation of Windows on ARM processors.
* Store. The mobile store was extended to support Windows 8 applications.
* Hardware. A new hardware business, delivering Intel x86 and ARM-based tablets.

These were all immensely ambitious efforts and to a large degree technically successful (more below on what did not go so well).

The final thread that I want to introduce is Chromebooks, released in June 2011, in the middle of the Windows 8 cycle. Networking has been a key part of computing since the earliest days and there have been lots of different system design points for how to distribute computation between devices on the network. With the rise of the web browser as a strategic target platform, Google made the decision to ship a device (and allow other manufacturers to ship devices based on the design) that were essentially just made to run the Chrome browser. The thing that enabled these devices to gain in popularity compared to other earlier attempts at “network computers” was the reliability (directly because of the simplicity), the low cost of manageability in a multi-user environment, existing wireless infrastructure and most importantly, the apps.

Google Apps had been growing in popularity in schools and here you had a device that was custom designed to run these applictions. A student could log into a device (which they already needed to be able to do to use Google Docs from a PC or Mac) picked off a cart of machines and their environment would fault in gracefully. They could be up and running with their documents on a completely new machine in seconds. This speed and reliability was critical for shared classroom use.

Note that like the situation with the iPad, you had hardware, OS software and an application ecosystem all aligned to deliver a great experience for a specific scenario. This hit a different suite of features than tablets since it targeted an environment where devices were swapped between users and that were still text-heavy, making a device that fully supported multiple users gracefully and with integrated keyboard a better choice.

So what happened with this first generation of Windows on ARM devices? Why did they fail to take off?

Obviously saying why a product was *not* bought is always an exercise in story-telling. One important factor was the resiliency of the laptop form factor. Despite some high-visibility experiments in a few school systems and large consulting organizations where iPads were deployed widely, in fact laptops have held their own as productivity devices (whether PCs, Macs or Chromebooks). Even Microsoft’s successful Surface Pro (Intel-based) line of tablets are mostly used in a laptop form factor. This meant a “productivity tablet” was not a large distinct device category. The lack of a new device category also doomed efforts to kick-start the app ecosystem necessary to differentiate these devices.

As a pure tablet, the Windows ARM device had a hard time competing head-to-head with the iPad, which had a two year head start in the market and a vastly larger app catalogue. As a lightweight PC with touch (essentially the niche the Surface Pro filled), the lack of Win32 application compatibility left these Windows ARM devices as ugly ducklings. They were not distinct enough in size and battery life to define a new category and overcome those functional negatives. Creating a virtuous cycle of devices, users and developers is a tremendously challenging achievement that has only succeeded a few times in our industry.

So what has changed for this new generation of Windows ARM devices?

One thing that has *not* changed is that Windows is still trying to dig out of the ecosystem rot issues. These devices by default ship with Windows S, the version of Windows which only allows store apps to be installed. But Microsoft still needs to allow these machines to be upgraded to full Windows for free since there are still important Win32 apps that are not in the store.

Some important things have changed since Windows RT shipped. These devices can not only run Win32 applications, they can emulate x86 code so do not require the porting effort we went through to get Office running on the original Window RT device. Microsoft now allows Win32 apps in the Windows Store, so even devices running Windows S have access to Win32 applications (if the vendor has done the work required to move them into the store). In contrast to the rewrite required for WinRT applications, Win32 store applications get the install and updating benefits for a much more limited cost. So there is some carrot there for developers.

ARM processors have improved in performance and emulation appears efficient enough that Windows is using it for some of its own code and Office is following the emulation strategy rather than the native ARM compilation it used for Windows RT. That bodes well in general for the performance of these devices although we will have to see what the battery impact of emulation is. Battery life is being used to differentiate these devices so an approach that causes battery life to suffer would be counterproductive. Microsoft has also added OS capabilities to resource manage Win32 applications in order to mitigate bad behavior. These were developed during the Windows 8 timeframe and extended in Windows 10.

In the announcements, Microsoft is calling these “always-connected PCs”. Obviously, this category already exists in practice (you can buy a wide variety of PCs with integrated LTE) so in the short term these will show up as devices that offer a slightly different palette and trade-offs between cost / performance / battery life / weight / connectivity than traditional x86 PCs without really defining a new device category.

One aspect of the Chromebook competition is the low cost of Chromebook devices. Windows ARM devices could theoretically challenge this. Cost is a significant criteria, but the striking functional behavior of Chromebooks is how well they work as multi-user devices. This arises from the fact that the whole stack is designed as an asynchronously populated local cache of cloud data. The Windows PC stack tends to be heavier weight with the assumption that more data should be locally cached on the device (to better leverage local device capabilities) and the user is more willing to wait till this happens (since it only happens rarely as you start using a new device). This directly impacts how well these devices work for multiple users swapping out shared devices. This is a pervasive ecosystem issue so is not trivial to change.

Windows on ARM does give Microsoft optionality in more aggressively leveraging the ARM ecosystem for new devices. That optionality might be more important in the long term than any immediate breakthrough.

* [Microsoft](https://hackernoon.com/tagged/microsoft?source=post)
* [Microsoft Windows](https://hackernoon.com/tagged/microsoft-windows?source=post)
* [Arm Processors](https://hackernoon.com/tagged/arm-processors?source=post)
* [Why Windows On Arm](https://hackernoon.com/tagged/why-windows-on-arm?source=post)
* [Windows](https://hackernoon.com/tagged/windows?source=post)

One clap, two clap, three clap, forty?

By clapping more or less, you can signal to us which stories really stand out.

195

 Follow

[](https://hackernoon.com/%40terrycrowley?source=footer_card" \o "Go to the profile of Terry Crowley)

[**Terry Crowley**](https://hackernoon.com/%40terrycrowley)

Medium member since Mar 2017