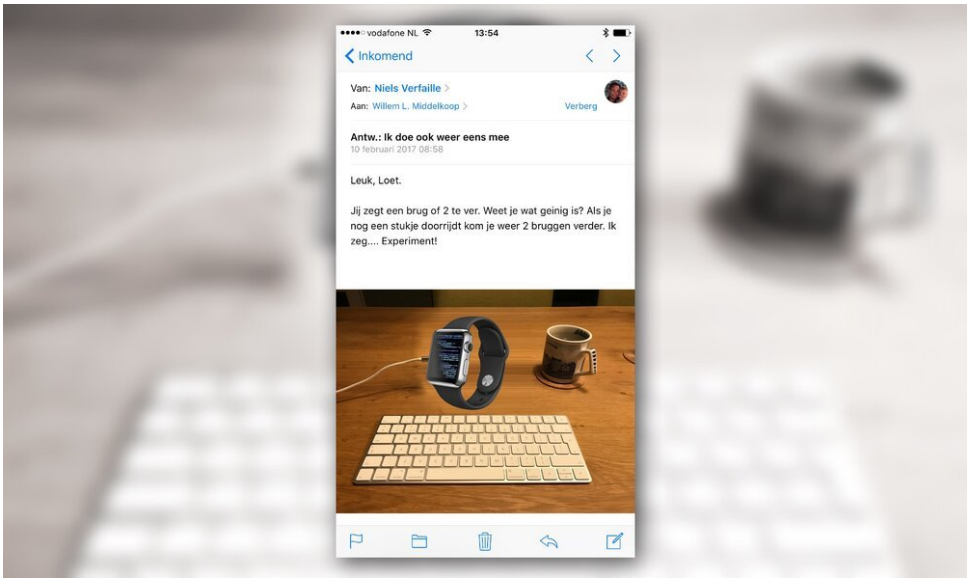
Programming on Apple Watch

Serious about crazy experiments

Willem L. Middelkoop
Feb. 16, 2017

Over the past years I have been no stranger to crazy experiments, but this time I really wanted to push it into the extreme: programming on an Apple Watch. Would it be possible to actually write code on such a tiny device? Why even bother? This post is about the case for crazy experiments, and why you should try too!

A week ago I was exchanging some emails with my buddy Niels, telling him about this idea of using an iPhone to get work done. He suggested (in Dutch) that I would take that idea to the next level... Apple Watch.



Message from my buddy Niels, suggesting I take my "working on a smartphone"-idea to the next level, complete with a Photoshop mockup.

While he used his brilliant Photoshop skills, I used Xcode, my Linux knowledge, SSH, VIM, the local network, a bluetooth keyboard, an actual Apple Watch, some time and patience.... resulting in this proof of concept:

And so I did! Programming on Apple Watch using VIM, SSH, a Bluetooth keyboard and coffee.

Real programming code, real Apple Watch. No Photoshop.

How it works (for dummies)

I created a small Watch app that connects to an external computer that runs the development software. The keyboard is connected to this external computer, so that I can actually write code. The principle is simple and can be compared to the Camera-app that is on the watch by default: you use the watch as a viewfinder for the camera in the iPhone. This Watch app is like a ”viewfinder” for programming code.

How it works (for those tech savvy among us)

In the local network I run a small NodeJS server on a Linux machine that takes screenshots from a shell session running my favourite programming code editor, VIM. The keyboard is connected to this laptop using Bluetooth. On the Watch I created a simple app that fetches a new screenshot every few seconds and use that to show the image full screen. It's a proof of concept, not anything worthy of the AppStore (there is quite some latency between refreshes - but it works). If you're thinking about building it yourself, look for *WKInterfaceImage*, *UIImage*, *NSURLSession* and *dataTaskWithURL*

The source of the Magic: my ThinkPad X1 connected to the local network.

Why

I experiment regulary, because it allows me to learn about the way I do things. The experiments force me to rethink things I take for granted or normal. In many ways **"the chase is better than the catch"**.

I actively experiment with new hard- and software, just to stay up-to-date and learn new things. In 2008, few months after the AppStore launched, I was among the very first to start with iOS development. Even though the potential was unclear, I learned and experimented with the technology. My early experience helped me program the Snake '97 app in 2011, another crazy idea that turned out to be a *20 million downloads*-game... :-))

Snake '97 - the original idea and stars of the game, the Nokia 5110 and 3310 - possible because of earlier experimentation with technology

What I learned

So instead of asking ”Why?” you'd better ask ”What did I learn?”. Well, quite a few things actually:

- **Move your programming environment into the terminal:** Go text based, ditch the IDE (like Visual Studio) and learn to properly use VIM (or emacs) with keyboard shortcuts. These tools run practically on *any* device and don't require a proprietary operating system.
- **Bye bye to the Mouse:** keep your fingers where you write code, on the keyboard. Instead of reaching for a mouse, use the keyboard (shortcuts). It may take some time getting to know them, but in the end you are *much* faster. In addition to speed you also gain the ability to use any screen, as more devices support keyboards but not mouse (think game consoles, smart TV's, AppleTV, old school devices)
- **Use the pixels, wisely:** If you constrain yourself to a small screen, you are forced to better think about how to use the pixels. Do you really need all those toolbars visible, all the time? The fewer interface elements permanently visible, the more space you'll get for the actual content. Less distraction, more focus.
- **Write better code:** To keep things readable on a small screen, you'll learn to really pay attention to things like code structure, function logic and naming. Small screens encourage shorter lines of code and more compact function bodies. It forces you to make things simpler and shorter, repeat yourself less. These advantages translate to bigger screens too.

- **You don't really need a PC anymore:** If it's possible on a Watch, think about those other devices you carry around, like your smartphone and tablet. Being able to better use your mobile device(s), will make you less dependent on traditional PC's.
- **Apple Watch has potential:** Earlier, Facebook developers already succeeded in getting the famous Doom 3D game running on Apple Watch. I learned, too, that the device has serious potential. I am beginning to think that the Apple Watch is much more an "always there, mobile computer" rather than something to compare to a traditional mechanical watch.

Feeling mobile: this blog post was created using an iPhone, a keyboard and some coffee! No PC!

Conclusion

This whole experiment really pushed me forward in terms of mobile computing. I learned to optimise my toolset and the way I work. In fact, this whole blog post was created on an iPhone, including the photos (taking them, cropping them), and it didn't feel weird at all. I love it because it makes everything easier, and that made this experiment worthwhile

Bonus: Nokia Communicator and Jolla

When I was writing this post, I wondered if my old Nokia Communicators still worked. After some fiddling with the old batteries I managed to get the E90 and 9300i working. Using Putty and 802.11b WiFi, I managed to connect to my newly created development system that I used for this experiment with Apple Watch. Guess what? It worked just fine - who needs a new phone anyway? :-)

Multi platform development done properly, Nokia Communicator E90 with Symbian series 60 from 2007, Nokia 9300i running Symbian series 80 from 2004, Jolla phone with SailfishOS with the funky other half keyboard (tohkbd), and the iPhone 7.

Nokia E90 Communicator running Putty on Symbian Series 60.

Nokia 9300i, officially not a communicator but nonetheless a device with advanced mobile communication options for its day (2007). Running Putty on Symbian Series 80.

Multi user, multi screen. Two Nokia Communicators (E90/9300i) connected to my work environment simultaneously using Putty.