How I Develop Things and Why

2013

5 min read • 1,119 words

Themes: Consciousness Technology Programming Spiritual Contemplative

I've always considered myself a bit of a software junkie. Nothing excites me more than a great piece of new software. Some of my best childhood memories are our trips to Grandma’s house, where I’d have access to a computer with a dial-up connection that I’d use to obtain freeware and shareware. I’d bring 4 or 5 floppies with me and try to cram all the games, waveform editors, and utilities that I could sneaker-net home.

Luckily today, excellent software written with passion oozes out of the app ecosystem. OS X and the App Store really fuel an economy of software built for humans by people that care.

Unfortunately, this doesn’t always hold true in developer software — text editors, modules, libraries, toolchains, &c. We are forced to deal with APIs on a daily basis that were not built with the user in mind. Over-engineering surrounds us as developers. Things that should be simple are often needlessly complex for the sake of being complex and “proper”.

Why should consumer apps and developer APIs be treated differently?

Have an Issue

The first step to developing something great is to have a real problem. You can’t solve a problem properly if you don’t experience it firsthand.

On the consumer app side of things, a great example of this is Microsoft OneNote. Have you used OneNote? It’s incredible.

Essentially, OneNote is a hierarchical freeform note-taking software that assumes nothing: you can type, use handwriting, embed files, cross-link notes, sync them online,&c.

Unfortunately, OneNote is only available on Windows. This kills me. I would love to think that Microsoft’s would port this lovely peice of software to OS X, but I doubt it will ever happen.

If I ever decide to actually ship a consumer product, it will be something akin to OneNote for OS X. It would be incredible. It may not be for many, but for people that resonate with my problem, it will work wonderfully. It would be a reaction to a real problem, not an engineered app an entrepreneur thinks will fill a gap so he can make some fast cash.

GitHub wasn't built for the developer community at large; the founders built GitHub for themselves

> Tom Preston-Werner, Chris Wanstrath, and PJ Hyett started GitHub in 2008 because they were frustrated with existing Git hosting solutions. Their 'scratching your own itch' approach became one of the most successful developer platforms, validating the principle that solving your own problems often creates broadly useful solutions.

. The problem they solved simply happened to resonate with millions of developers because they themselves happen to be developers.

37Signals didn't build Basecamp for a world full of project managers and consultants; they built it for themselves. They also developed Ruby on Rails for themselves, as Ruby developers that were repeating themselves too often.

How pragmatic.

These companies didn't need to commission lengthy case studies and perform market analysis. They didn't setup faux AdWords to measure the effectiveness of various marketing copy. Yet, they are astronomically successful. How is this possible? They know exactly what they want to build, how it should function, and how it should look because they were building it for themselves and not for others.

Let's go back to the developer's side of things.

A great example is my Requests module. I was a heavy user of Convore at the time, and I wanted to interface with it programmatically. So, I set out to build a Python module that wrapped the Convore HTTP API. Unfortunately, this was easier said than done. Dealing with Python's standard library for HTTP was a complete and total nightmare. It was over-engineered.

I love Python because it's a language designed for Humans. Why should modern HTTP be so difficult? So, I sat out to discover what it was that I wanted, and built exactly what I needed. It resonated well with others.

Nothing is more satisfying than using your own tools to Get Things Done.

Respond with a README

Before I start writing a single line of code, I write the README and fill it with usage examples

> Readme-Driven Development was popularized by GitHub co-founder Tom Preston-Werner in 2010. This approach forces developers to think from the user's perspective first, often resulting in cleaner, more intuitive APIs—exactly what made Requests so successful.

. I pretend that the module I want to build is already written and available, and I write some code with it.

This has an incredible effect: instead of engineering something that will only get the job done, you start to interact with the problem itself and build an interface that reacts to it.

You discover it. You respond to it.

Great sculptures aren't manufactured — they're discovered. The sculptor studies and listens to the slab of marble. He identifies with the stone. Then, he responds. He enables the marble to speak for itself, setting free something beautiful that hidden was inside all along.

He responds.

This is what responsive design is all about. It's not merely a method to engineer a web design that will function on a phone, tablet, and desktop.

Beware lest you lose the substance by grasping at the shadow. Responsive design is about making a design that identifies and understands itself enough to respond to the environment it's placed in. It is about setting your design free from arbitrary constraints. It is setting free something beautiful that was inside all along.

The sculptural metaphor here reflects what I would later recognize as fundamentally contemplative—the same kind of patient attention and responsive creation that characterizes programming as spiritual practice. Whether working with marble, code, or consciousness, the process is one of deep listening and conscious response.

This is known as Readme-Driven Development. I call it **Responsive API Design**.

Build

Now that you know what your API is: Build it. Make it happen. If there's a significant amount of complexity behind a simple call, make a layered API: a porcelain interface that sits on top of a verbose API that sits on top of an low level integration interface.

The user API is all that matters. Everything else is secondary.

Once your software is released, improve it! Add new features, better security, optimal performance, and rigidity. But never compromise the API.

Manifesto

Build things that you want. Build things that you need. Build things for you.

The Golden Rule™:

> Do unto others as you would have them do to you.

Adapted to:

> Build tools for others that you want to be built for you.

> This ethical foundation—treating users with the same respect and care you'd want from the tools you use—would eventually expand into a comprehensive approach to conscious technology development that recognizes every line of code as an act of service to human flourishing.