What follows is an email I wrote and distributed to the network and systems engineers at one of my employers. Please use this document to learn about my writing style, my approach to programming and automation, and how much respect I have for my coworkers.

In late 2019, Cliff Kuang wrote an article for Wired magazine entitled '[How the Dumb Design of a WWII Plane Led to the Macintosh](https://www.wired.com/story/how-dumb-design-wwii-plane-led-macintosh/)'.  I am not a fan of Macintosh computers, but I am a fan of WWII history, and have a deep respect for how Apple managed to produce several products that "just work".  I read this article and walked away thinking "I can be like that.  That is who I want to be at work."

I invite you to read the entire article, but in case you don't, here are the important parts that you'll need so the rest of my email makes sense.  It's a lot to read, but the language is easily digestible.

* For all the triumph of America’s new planes during World War II, a silent reaper stalked the battlefield: accidental deaths and mysterious crashes that no amount of training ever seemed to fix. And it wasn’t until the end of the war that the Air Force finally resolved to figure out what had happened.
* The Air Force called upon a young psychologist (Paul Fitts) at the Aero Medical Laboratory at Wright-Patterson Air Force Base near Dayton, Ohio.  Fitts was supposed to know how people think. But his true talent is to realize that he doesn’t.
* When the thousands of reports about plane crashes landed on Fitts’s desk, he could have easily looked at them and concluded that they were all the pilot’s fault—that these fools should have never been flying at all. That conclusion would have been in keeping with the times. The examples slid back and forth on a scale of tragedy to tragicomic: pilots who slammed their planes into the ground after misreading a dial; pilots who fell from the sky never knowing which direction was up; the pilots of B-17s who came in for smooth landings and yet somehow never deployed their landing gear. And others still, who got trapped in a maze of absurdity, like the one who, having jumped into a brand-new plane during a bombing raid by the Japanese, found the instruments completely rearranged. Sweaty with stress, unable to think of anything else to do, he simply ran the plane up and down the runway until the attack ended.
* But the culprit was maddeningly obvious for anyone with the patience to look. Fitts' colleague Alfonse Chapanis did the looking. When he started investigating the airplanes themselves, talking to people about them, sitting in the cockpits, he didn’t see evidence of poor training. He saw, instead, the impossibility of flying these planes at all. Instead of “pilot error,” he saw what he called, for the first time, “designer error.”  The reason why all those pilots were crashing when their B-17s were easing into a landing was that the flaps and landing gear controls looked exactly the same. The pilots were simply reaching for the landing gear, thinking they were ready to land. And instead, they were pulling the wing flaps, slowing their descent, and driving their planes into the ground with the landing gear still tucked in.
* Chapanis came up with an ingenious solution: He created a system of distinctively shaped knobs and levers that made it easy to distinguish all the controls of the plane merely by feel, so that there’s no chance of confusion even if you’re flying in the dark.  That ingenious bit of design—known as shape coding—still governs landing gear and wing flaps in every airplane today. And the underlying idea is all around you: It’s why the buttons on your videogame controller are differently shaped, with subtle texture differences so you can tell which is which. It’s why the dials and knobs in your car are all slightly different, depending on what they do. And it’s the reason your virtual buttons on your smartphone adhere to a pattern language.

...This (below) is the most important quote, so pay attention...

* They realized that it was absurd to train people to operate a machine and assume the human would act perfectly under all conditions.  Instead, designing better machines meant figuring how people acted without thinking, in the fog of everyday life, which might never be perfect. You couldn’t assume humans to be perfectly rational sponges for training. You had to take them as they were: distracted, confused, irrational under duress. Only by imagining them at their most limited could you design machines that wouldn’t fail them.  Along the long and winding path to a user-friendly world, Fitts and Chapanis laid the most important brick. They realized that as much as humans might learn, they would always be prone to err—and they inevitably brought presuppositions about how things should work to everything they used. In some sense, our limitations and preconceptions are what it means to be human—and only by understanding those presumptions could you design a better world.

Chris brought me in to be a DevOps Architect to create solutions for Infrastructure-as-Code, with the end result being a better work day for you.  I know that working a complicated change at 11 PM after a full day of work and family life is tough, because I've been there.  I know that being woken up and told to jump on a P1 Incident call at 3 AM because you are the Subject Matter Expert is an experience many of us have shared, and I know that during these times the last thing you want to do is try to debug code that went wrong.

To that end, I want to work with each of you to understand how you use code in your job.  Where do you store your code?  How do you collaborate on code with others?  What development tools do you use?  How do you test before pushing it into production?  How do you run it in production? Does your code collect data and churn out a report?  Or does it actually effect a change?  Perhaps you hate a particular script but there is no better alternative?  Are the command line arguments inconsistent or confusing?  Is there little to no documentation?  Does it fail randomly?  Do you wish it did more?  These are only a small sample of the questions I have for you.

Once I have digested all that you have to show me, Jared and I will begin working on ways to bring your code under source code control for version tracking, documentation, diagnostics, and integration testing; all of which will help a new person get up to speed using/supporting your contributions [should you ever get hit by a bus](https://www.youtube.com/watch?v=_fiQ76CXkeg).  We will also investigate centralized automated deployment, centralized data collection, and credential management.  I will help you every step of the way by helping you write documentation, providing group trainings or one-on-one sessions, and listening to and using your feedback.  I will work with you to improve your code so it can perform the 11 PM change without you, and so you'll never have to debug code at 3 AM.

My personal promise to you:

I will never steal your thunder.  I will always give you credit.  I will set you up for success.

My request of all of you:

Please schedule some time with me over the next few weeks to walk me through how and where you use code in your job.  My calendar is wide open and I am in the Hamilton office from 7 AM to 4 PM.

- Greg DeCecco