SSID: NGGuest

Password: welcome88

MacDMV Presents

Testing Sierra & iOS 10

Tom Bridge
Technolutionary LLC

I think it's important to recognize a couple things about our current situation as admins.

1) It's never been this good to be a Mac Admin.
2) But it's also never had quite so many weird quirks and challenges

So, with those things said, let's go talk about where we're going and how we get there.

Why Do We Test?

Why Bother Using an Unfinished, Unrefined Operating System?

When the tickets are coming hard & fast, and we're focused on the day to day, is there even time to test? We'll get there as things get closer, right?

And then a month slides by and we're at beta 3.

And then another month slides by during summer break with vacation schedules and suddenly it's mid September and beta 5 is out.

And then it's release day.

Crap. Does it work?!

So We Know What's Broken

We Test so we know what's broken. We test so we're prepared when someone updates on day 1. We need to know what's going to happen when someone installs Sierra or iOS 10 on their devices, because we have to support them.

The fix can be "we're going to roll you back," but you have to be ready for the fallout that comes with that. Some users you may have the political capital to rollback. Some you won't.

If you find things that are broken, you can warn your users well in advance that there are broken things that aren't worth risking. It's not going to be a picnic on Day One, but it's the cycle weren't aware of.

If we don't know what's broken, we're working with an incomplete picture of our future. That's unwise for a number of reasons.

Job 1: Build A Test Platform

- Setup a VM or Loaner with **actual, useful data.**
- **Live** in that VM or Loaner while testing as much as you can stand to do so.
- Keep that VM or Loaner **in line** with your existing environment.

So We Know How New Things Work

The other important reason to test is that you need to know how the new features are going to work.

How does Siri fare on the Mac? Is it a Data Hog? Does your environment make Siri verboten? Do you have a plan for how to disable it if it is necessary?

How does the new Apple Watch Unlock work? Is that going to pass muster with your security group?

Who here is scared of your users turning on iCloud Drive for their desktop and documents folder?

Is your MDM ready to work with iOS 10? Who's checked?

What are the consequences of iOS 10's changes to the lock screen for your organization?

So We Know How New Things Fail

More importantly, though, it’s good to know how these things are going to break, so that we can troubleshoot them. Maybe C-levels and Directors don’t need to know the ins and outs of it, but your techs absolutely do.

Job 2: Build a Test Plan

- Construct a list of all your critical elements
- Test them with each build
- Catalog results and build checklists

Building a list of your critical dependencies is a large part of this testing process. Every piece of your infrastructure is a part of that list, to varying degrees. Network access (802.1X, Wi-Fi), Deployment cycle, Patch Management, Profile Management, Printing (yes, printing!), Intranet Access, Directory Bindings, LDAP Lookups if necessary, Exchange access.

Build a hierarchical list of things to try so that you can stop your testing when it gets to a kill point for your environment, so you don't spend time mucking around with the Intranet on Safari if your AD Lookups are failing.

Build that list and test it with every build that comes out. Note what's wrong and log it for your reference, and so you can…

But most importantly, we test so we can write good feedback.

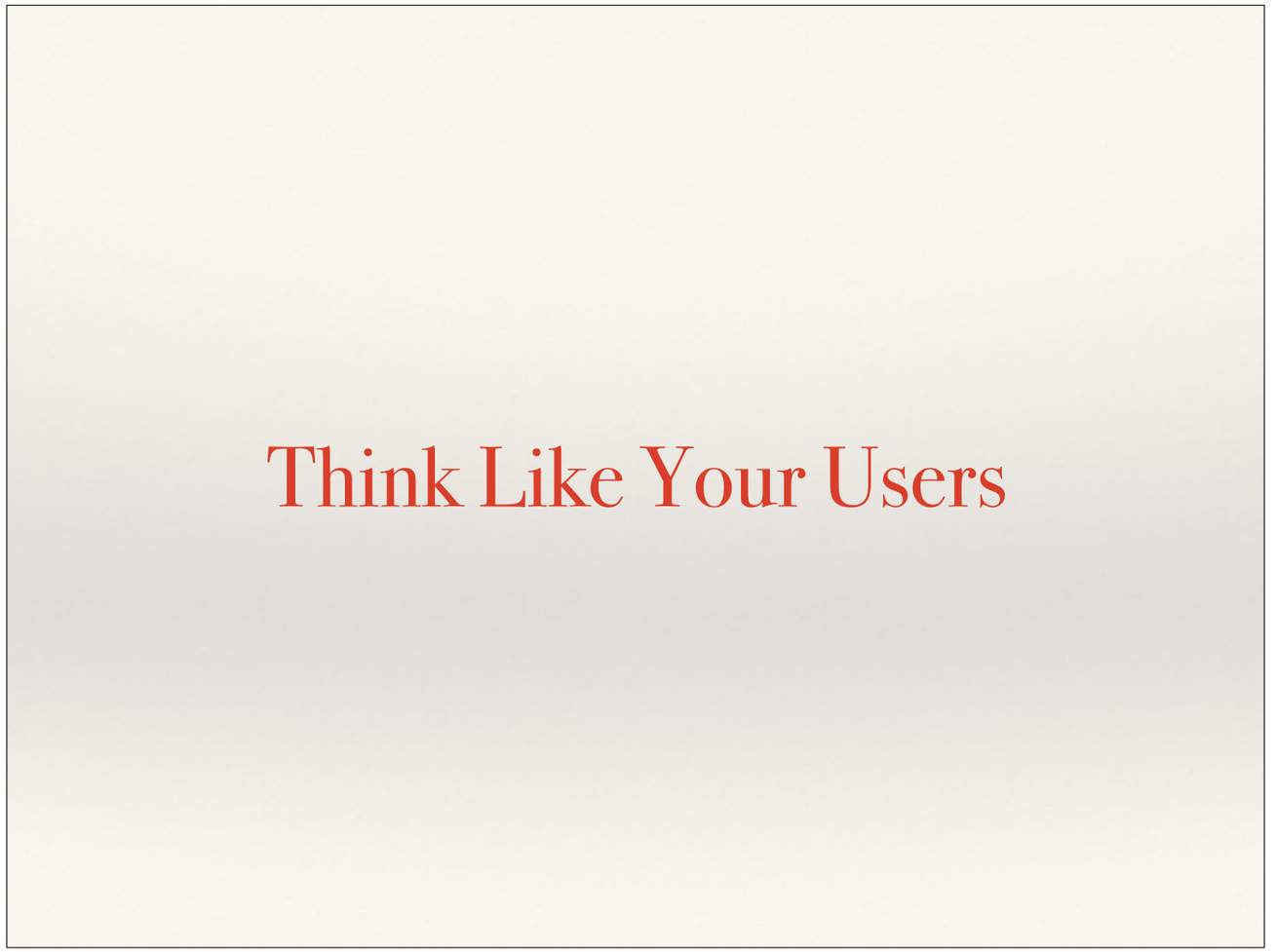
Feedback, like tech support, is an art form best perfected through practice.

Job 3: Write Good Radars

- Concisely Explains the Problem
- Follows the “I expected X, but Y happened” form
- Suggests an Alternative Behavior, if necessary
- Demonstrates Potential Impact (# of machines, users)
- Is Devoid of Emotional Baggage

Good Radars provide information to engineers in a clear and concise manner. Some of these are going to be hard to write for a couple reasons:

1. Some of these are problems that are repeated from previous versions that haven’t been addressed yet.
2. Some of these are problems that need detailed regression in your testing environment, and behavior may vary between Virtual Machines and Physical Hardware installs. This is where those notes from Job 2 come in very, very handy.
3. Now, you may have an Alternative Behavior Request for a given situation. If it’s outside the realm of the usual (ie, I don’t want this to crash), you might not get what you want. But if you request it, come at it from the point of Apple’s Human Behavior Interaction Guide, which I’ll link in my blog post about this.
4. Next: Show Apple why you care. Show them how many machines it will affect in your org. If you’re small, get friends and colleagues, here or in Slack or elsewhere, to file duplicate reports stating how much they will be affected by the problem.
5. Lastly and this is important: Keep it professional. The engineers do not care how many times you have had to write this out. They do not care how many different versions of this you’ve had to deal with. Keep it 100 with them.



No, I’m not going to make a joke about giving yourself a concussion first. But come to the testing process with your users’ expectations, not just your own. Don’t cut yourself slack. Get impatient.