10 lessons for future big bets from our rider app redesign, Project X

Sravanthi Kadali

This is the first in a three-part series. Part 2 is on user insights from the 54 A/B tests we ran within X to isolate changes and additional A/B tests by the Request XP. Part 3 will be on options for measuring improvements over time in long-running experiments (in Q1, with Experimentation).

As we reflect on learnings from 2018 and three months after the roll-out of our new rider app to 100%, I want to share 10 takeaways that are relevant to other big bets (including but not only redesigns).

As Nik shared several months ago and as Martin said in his initial pitch for the project, Project X was an opportunity for Lyft as a whole to learn how to roll out a big bet. The team has had multiple retros: a retro with the core project team and partner teams as of February 2018, a retro on GTM in June, and two retros in July, one with partner teams and one with the core project team from March through August. **The conclusions in this post represent only my own views, but were informed by what the X team and partner teams voiced in those retros.** I'm hoping that we'll be able to keep investing in ambitious projects while improving our process for future ones.

For an excellent write-up about redesigns based on experiences outside of Lyft, also see this Medium post by Eran.

TL;DR

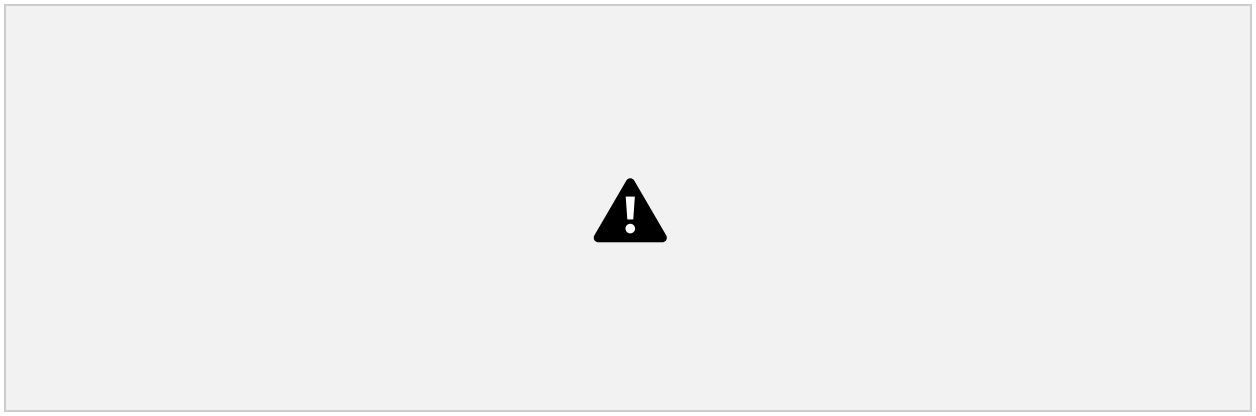
> Get organizational alignment on the "why." An exec sponsor is critical.
> Narrow in on one clear goal, or sequence your goals.
> Get organizational alignment on what this means for other teams. An exec sponsor is even more critical here.
> Partner with other teams, and communicate regularly.

Do you maintain two code bases, and where do other teams build? (If your goal is user experience and this may not roll out, ask teams to build on the old code base.)
How much do you test incrementally? (You need to test as many changes incrementally as you can.)
How do you figure out how to improve? (Focus on qualitative feedback.)
Obsess over bug-fixing.
Expect a long period of iteration.
Expect to need a lot of people, and protect that team's sanity.

What is Project X?

Project X is the rider app redesign that was rolled out to 100% in September. There were three major phases of the project: early explorations between January and November 2017; initial external testing from November 2017 through February 2018; and roll-out of the app from 5% to 100% of users from March 2018 through September 2018. The final version of X increased rides by +4% for iOS users who onboarded on X (were new when they first saw X), when measured over 2 months.

The "X team" was a group of individuals that are now in Mobile Infrastructure, Core Design, and Core Passenger. Many members of the team changed in March 2018, when the project was "rebooted" (it's worth mentioning that the fixes that led to rides gains for new users happened *prior* to this team change). My own involvement was from August 2017 through September 2018, initially as the project's Data Scientist (back then, "Data Analyst").



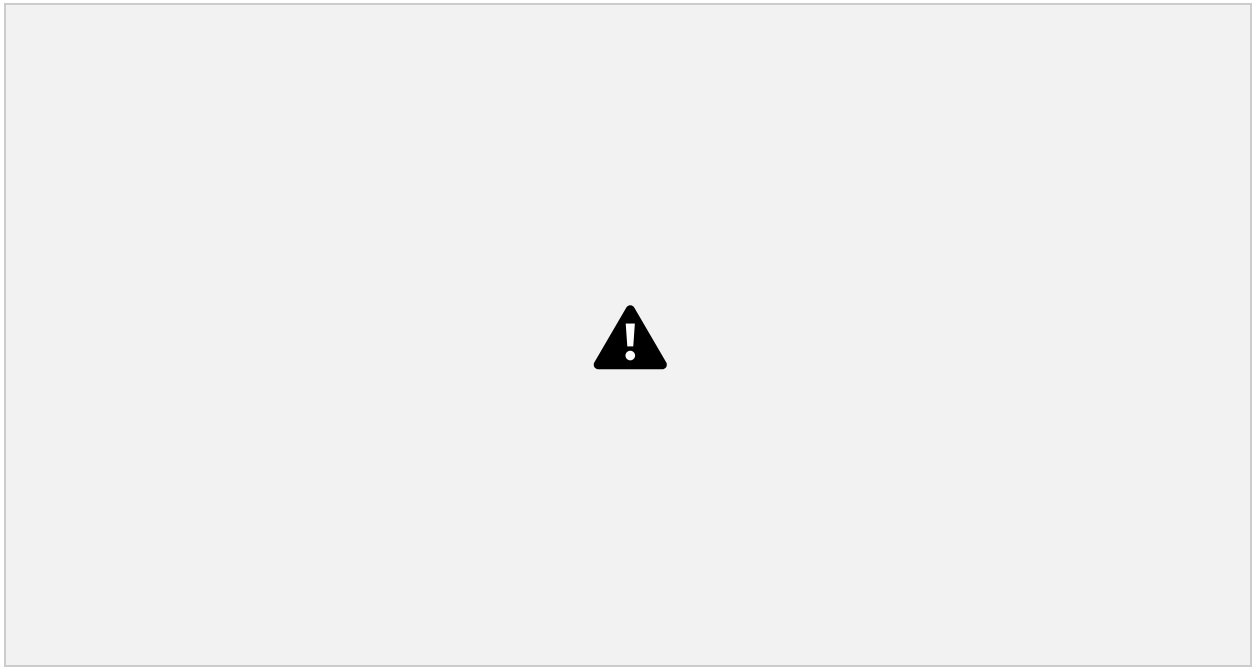
click to add a caption

Thank you to many people for reading prior drafts of this post and providing helpful feedback: Dave Bellona, Katie Dill, Gordon Tindall, Derek Salama, Eran Davidov, Kevin Wang, Martin Conte Mac Donell, Jeff Hurray, Alan Fineberg, Dave Kim, Nadia Eldeib, Jacquelin Hansel, Thibault Martin, Alexis Baird, Akshay Patil, Lily Sierra, Sachin Agarwal, and Jimmy Young Young. And

thanks to Gordon Tindall, Dave Bellona, and Marc Haumann for many of the images used here.

#1: **Get organizational alignment on the "why." An exec sponsor is critical.**

Project X began as a blue-sky design project. The initial team set out to rethink the Lyft experience, exploring divergent ideas like using a bracelet to request a Lyft. I wasn't around for this initial phase, and by the time I joined the project, the exec team had greenlit and set an aggressive deadline for "shipping X," with guidance to release the entire new app without A/B testing.



click to add a caption

However, what “X” represented had quickly changed: through design reviews, the team tweaked the original design, and by the time it was shared with the rest of the company in August and launched to users in November 2017, “X” was less divergent and instead represented a change the user flow to request and complete a ride and a new, more modern design language. Since this now represented a UX and UI redesign, we were asked to A/B test X against the current app to understand if it was better for users (which in my opinion was the right thing to do). The team was also signed up for a new goal to increase rides by 1%, weeks before external launch. Both represented shifts in goals for the original project team.

When goals shift significantly, raise alarms and evaluate whether your product still meets the new goal and is something the team and stakeholders believe in. There are many good reasons to take a big bet, but you need to know what your reason is--and make sure that your stakeholders agree. This will inform every other decision you make.

Big projects also need one exec sponsor from the start, which could’ve helped in this early phase. Through February 2018, there was no single decision-maker at the executive level. Having an executive sponsor from March onwards in Katie was invaluable in many ways, including shielding the team from churn in the months before 100% roll-out and focusing us on the singular goal of delivering a better user experience.

#2: **Narrow in on one clear goal, or sequence your goals.**

By November, the X app represented a new visual aesthetic to rebrand the Lyft app and build scalable UI components (our design system, the Lyft Product Language, was created in tandem with Project X); a technical re-architecting of the Lyft app; and a new user flow, including a destination-first request flow, easier mode selection and in-ride panels, to serve both user needs (a more focused request flow) and internal needs (easier addition of new ride types).

To me, X now did not represent a "big bet," but rather several big bets in one. Pick one goal. With multiple goals, it was difficult to make trade-offs and know how to measure success.

Examples of a more focused (but still ambitious) goal that the project could have focused on:

> User goal & internal business goals: "enable users to choose the right ride type for getting where they need while making it easier to add new ride types and collecting more destination information to link rides"; or
> User goal & internal business goals: "provide more relevant information in-ride"; or
> Internal technical goal: "re-engineer the Lyft app from scratch to be more scalable"; or
> Internal design language goal & external brand goals: "create a reusable UI framework while modernizing the look and feel of the Lyft app"

In reality, as above, some of these goals might need to be paired; but if that's the case, then you should aim to sequence which goal you tackle first when it's time to build and test. Whatever the goal is, don't change *anything* that isn't in service of that primary goal. For example, the version of X as of November 2017 had various changes to the post-request flows. When cancellations initially went up on X relative to non-X, it was impossible to know whether this was due to changes in the post-request experience or due to fundamental issues in the request flow.

#3: **Get organizational alignment on what this means for other teams. An exec sponsor is even more critical here.**

Large-scale redesigns are often a company-wide effort. Even if you do your best to avoid collisions, many teams beyond the core team will be affected as the new design language rolls out. You may have two code bases at once, adding a tax to the company.

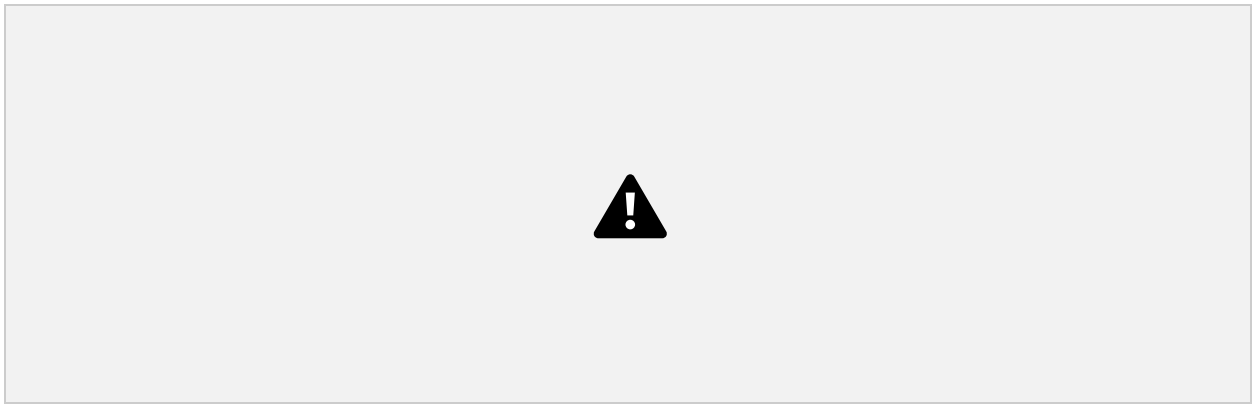
Again, you need an exec sponsor upfront for any major redesign: to get exec alignment on how much the company is willing to invest, in order to then align resources and teams, and aid in communication top/down and laterally. If this is going to be a focus for the company, be honest about the amount of resources it will take, as Lyft did with decomp.

#4: **Partner with other teams, and communicate regularly.**

X was a secret project until August 2017. After it was public, the team was strapped for time and lacked a TPM to help with regular communication. In the absence of communication, there was misinformation about the project and organizational mistrust worsened, which was a bad outcome for everyone.

Two things worked well after March 2018, led by Johnson as the project's TPM:

- Weekly partner team meeting to discuss upcoming features in the passenger app, where we identified and resolved feature collisions and also shared insights on what teams were learning
- Weekly all-tech emails on progress



Weekly emails sent from March through August

#5: **Do you maintain two code bases, and where do other teams build? (If your goal is user experience and this may not roll out, ask teams to build on the old code base.)**

You need to figure out your build, rollout and testing strategy early. You have three related questions to answer:

(a) Do you maintain two code bases (which will enable you to A/B test the changes), or roll out the new changes right away?

(b) Do you ask the entire company to work on the new code base, or do you ask other teams to build on the “old” code base?

How much do you test incrementally?

The answer to the first question depends on the goal. For a *user experience* redesign -- meaning a redesign of a core user flow -- presumably the goal is a better user experience and success is inherently risky, because your solution could be wrong. For those two reasons, I believe the best answers to the first question are (a) maintaining two code bases in order to A/B test and (b) asking teams to build on the *old* code base. This answer might be different if your goal is to solve for internal needs.

Why?

- You need to understand the impact of your holistic changes to users, if your goal is to deliver a better user experience.
- You’ll reduce the pressure on your team to make the app work (which was felt in X). It isn’t realistic to assume that the project will succeed, or that it can be turned around in a short amount of time.
- If teams build on the old code base, they’ll be able to test new features with more sample (which was a problem for partner teams during X).
- You won’t introduce variability in the experience that your team doesn’t fully understand (which was a problem during X).

But if you ask teams to build on the old code base, you will need to plan for one important issue: the old code base will naturally get better. In the case of X, the team initially asked partner teams to build on X because we expected that improvements to the old code base would make it hard to measure results, and that problem did occur: high-impact projects launched on non-X between November and February, making our “control” group a moving goal post.

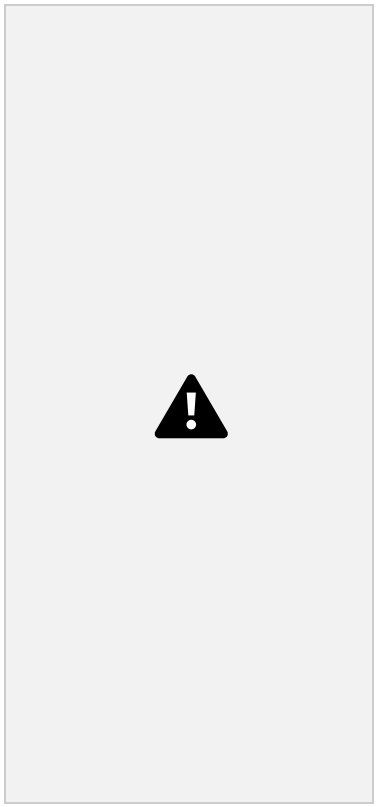
In future projects, there are two steps we might be able to take to enable teams to build on the old code base:

- Design a “clean control” group for the X experiment that is held out from the changes happening in non-X, similar to a global holdout; the Experimentation team has built an experiment exclusion tool that can enable this for future experiment
- Keep track of improvements happening to the old code base through regular communication with partner teams, and when something significant ships, port it to the new code base.

I’ve been asked whether testing parts of X incrementally could’ve also avoided the big fork between X and non-X. If X had focused on the technical goal or the new design language, then we likely would not have avoided a fork at some point. If X had focused on the “user experience” goal, that would have limited scope, which would have certainly helped; but given how many teams build on top of the request flow, any change will have collisions that need to be handled by deciding whether other teams build on top of your feature/product or separately. What we could’ve avoided was asking teams to build on the new branch rather than the old one.

#6: **How much do you test incrementally? (You need to test as many changes incrementally as you can.)**

Start with a design North Star that represents the best holistic solution, but build and launch toward it with careful steps and validation of smaller hypotheses.

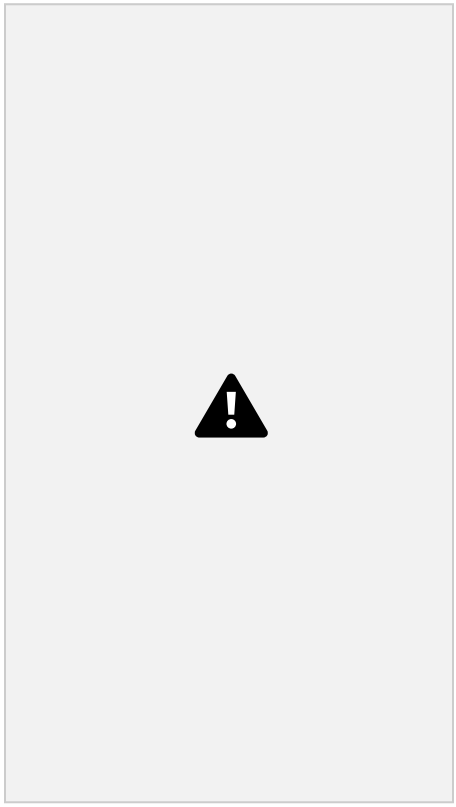


An older version of the mode selector on X

You *need* to test as many changes incrementally as you can without compromising the success of the overall design, for one primary reason: If you

make large changes and A/B test them, they'll most likely fail and you won't know why. (And even if you succeed, you wouldn't have learnings to share for future product improvements.) We learned this the hard way, even though we were warned. When X was launched externally to 1% of users, rides went down -2%, and it took a lot of effort to figure out why.

The X team ended up using A/B tests *within X* to isolate changes, which worked. The team ran a total of 54 experiments from January through August 2018; of these, 39 were rolled out, including many experiences that continued to be iterated on, and 15 were rolled back. Part 2 will have a full list of insights from experiments within X.



A more recent version of the mode selector

On the other hand, there are two benefits to bigger changes. Here's what you should consider:

(1) You might believe certain changes are only effective together. An alternative to running A/B tests within X that I'll use in the future is one multi-variant experiment at the outset. One of the components of X that I'm proudest of is the new mode selector, which allows users to compare time and price across multiple options. The new design language was critical to doing this effectively; interestingly, two similar experiments on X and non-X showed different results: when the concept of showing two modes to the user was tested on non-X, results suggested that one mode was preferred to two; later, we saw the opposite on X. If the goal of X had been to test a new mode selector *in a new design language*, we could have instead designed one multi-variant test where some users got a reskin and others got a reskin plus the new mode selector. This would've still been a big effort, but one with a more focused goal and a higher likelihood of yielding learnings

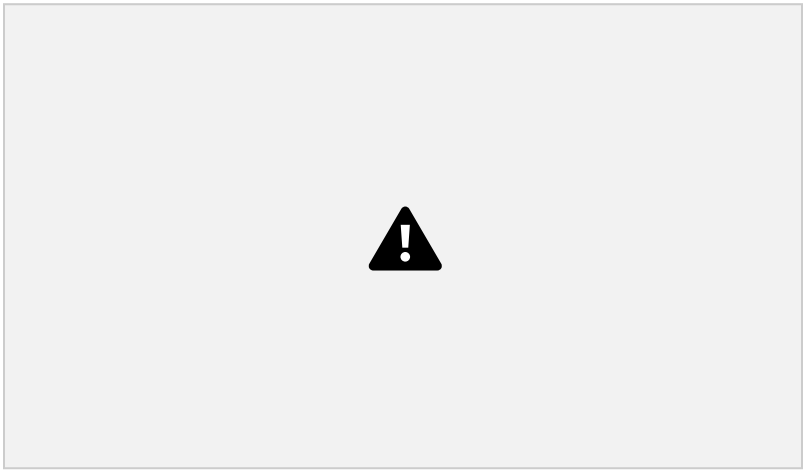
(2) If a goal is brand-related, sometimes you might want to make a splash. It's important that the team discuss this at the outset: if you do want to market the changes as one unified app, how can you do that while also mitigating the risk of shipping multiple changes at once? Which aspects can you A/B test to learn in the meantime? And finally, define what the "visual quality bar" is for the final version you go to market with (allocate time for this; we ended up spending the last few cycles before the 100% launch on fixing visual bugs).

#7: How do you figure out how to improve? (Focus on qualitative feedback.)

When X was launched externally to 1% of users, rides went down -2%. In Katie's words, "It is very common for metrics to go down when you change habit-based products. Don't lose hope. Give it time. do research. learn. Iterate. But don't be afraid to try new things." It's common for metrics to go down when you change a habit-based product, and there are also likely critical bugs you need to find.

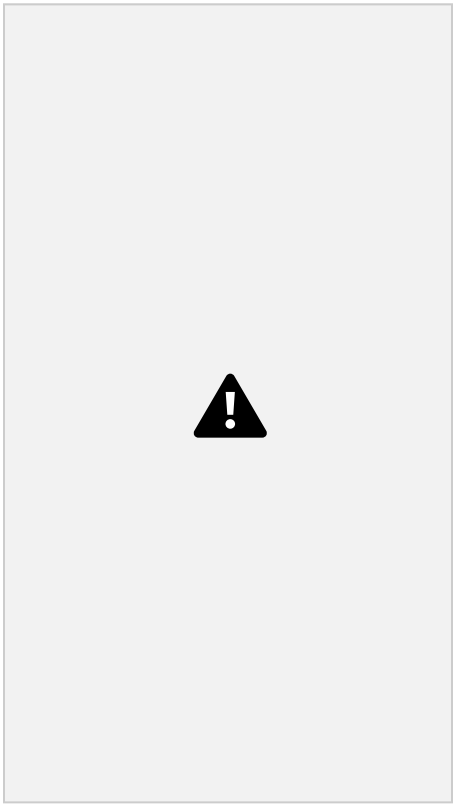
Even if you test as many variants as you can, you'll need a plan to figure out how to improve. You have two traditional research tools that are foundational:

- Qualitative UX research (see research conducted during X here, here and here)
- Data science (see the hypotheses we investigated)



click to add a caption

These are on the same spectrum, with one giving you more breadth and the other more depth. For something brand new, you likely need depth. You've ideally done user research on both low-fidelity prototypes (to validate concepts) and high-fidelity prototypes (for usability) before building anything, but even in experimentation phase, you should make sure you're staffed with 1 full-time UX Researcher and spend significant time getting qualitative feedback, which we didn't do enough of.



The version of Confirm Pickup in app version 5.16
that increased rides.

You also have two other channels to supplement on the “depth” side:

- Employee feedback
- A trusted tester program with external users

Feedback from employees who weren’t close to the project was a helpful signal, though it can’t replace user research. For example, I heard multiple reports that on the new Mode Selector and Confirm Pickup screens, users didn’t know they could tap to set pickup and dropoff. In app version 5.16, the primary change we made was aggressively fixing this issue with pins that said “Edit.” This was the first version that showed rides gains for new users.

The fix illustrated two things: (1) Users might not understand aspects of your product that you find obvious. (2) When you want to validate or fix an issue, you should try testing the fix in the most obvious or explicit way. We were able later on to replace the design with the Trip Bar to solve other usability issues posed by the larger map pins.

We also set up an external beta program for “trusted testers,” and would have benefited from expanding this program and surveying these users more regularly,

which would have also been a way to ease existing users into the new app. In the words of Dave, “Make sure users are fully aware of changes and bought in before it’s rolled out “

#8: Obsess over bug-fixing.

In addition to figuring out which features aren’t working, allocate a lot of time to fixing bugs. In the phase of the project from March through August, the team spent the majority of its time fixing bugs, fixing well over 1,000 bugs. We also didn’t wait to quantify the impact of bugs; we just fixed them.

This likely contributed to the improvement in existing user rides from -2% to neutral (we started a new experiment for existing users, so it’s impossible to know). It’s better to delay launches and spend more time in alpha to catch bugs than to rush the process and launch features with bugs. In Kevin’s words, “the EM must also be fanatical about bug triage and resolution and not let any bugs go untriaged and unprioritized.”

#9: Expect a long period of iteration.

The project will take longer than you think. Always plan for the worst-case scenario.

The client release cycle takes a couple weeks, and before that, you need to allocate time for analysis, design, research, and, of course, building the feature—and this feels even longer if you’re in a hurry to push out improvements and see the benefits.

Major projects need time, and this time is mostly incompressible. In Kevin’s words, “This was true with X and it was true with decomp. Both ran more than 2x over their initial estimates. Let's learn from these data points and be honest with ourselves when doing future timeline estimations. More engineers will not make this go twice as fast, although they will help increase the probability of success by fixing bugs. The reason that this time is incompressible is that it takes time to learn and iterate, and there's little you can do to expedite that process.”

#10: Expect to need a lot of people, and protect that team's sanity.

Starting in November 2017, we had limited time to figure out how the product was performing, make improvements and communicate them to the org. The original X team was surprisingly small (and scrappy!); we ended up giving up nights and weekends to make improvements to X, and were able to turn the app around for new users before the team change. The final team working on X also put in long hours, many of them in New York and without enough context shared on the changes that were being asked of them each week.

Long hours and lost personal lives are, in the words of Alan (who didn't work on X), "an indication that the process has failed, not that we should expect these sorts of heroics when squeezed for time." And given the timing of the team change, the original team working on X didn't get deserved credit for those efforts and results.

If you want to do all of the above, you need more people than you think--which is why you need to align on an organizational commitment up-front. In March's team shift, we had more dedicated people working on these areas, which enabled us to get across the finish line:

- TPM (throughout)
- UX Research (in both the pre-build phase and the iteration phase)
- Marketing (one note: in addition to all of the above where marketing was critical, for GTM you should budget time for compiling screenshots and coordinating across teams)
- Science (we needed 2 Data Scientists, not 1)
- Product (with 2 people from March to August, we were both still very busy)