Delighting our Customers: Innovation and Experimentation at Optimizely

DAE-HO CHUNG, Senior TPM, Engineering Operations, Optimizely
KEITH NOTTONSON, Senior Director, Product Operations, Optimizely
JEFFREY SING, Software Quality Engineering Manager, Optimizely

No matter how much data analysis, user feedback and competitive research you conduct, there's always a chance you will get it wrong. And getting it wrong comes in two forms: building the wrong product and building the product wrong. To avoid these outcomes Optimizely uses an experiment-driven product development cycle. Experiments are run during all the different stages of their product development lifecycle. Learn how Optimizely uses experimentation throughout their entire product development lifecycle, and the different ways they encourage and support innovation.

1. INTRODUCTION

On the Iron Chef, a cooking competition show, contestants try to make the best dishes with specific ingredients, and the judges grade and pass judgment on the steaming dishes. The criteria the judges look at—taste, plating (appearance) and originality—are very similar to how we innovate and experiment at Optimizely, and how we determine the success of our product and feature launches.

The Iron Chef competition requires the contestant to build a masterful course using a surprise ingredient. The first thing they are judged on is did they execute creating their dishes using this mystery recipe? How successful did the chef successfully accomplish their recipe? This is very similar to our question, "Did we build the right product?" Second, the chef's creations are judged on taste. Does their dish please the taste buds and do you want to eat more? For every product we build, we ask ourselves at Optimizely, "Did we build the product right?" Finally the last component the judges look for is appearance. Does the dish created look pleasing to the eye and appetizing? For every product we release here, we ask, "Did we get our customers to use our product?"

By using different experimentation techniques throughout the software development lifecycle from idea and discovery through to development and launch, we want to be confident that the "dishes" we serve to our users are not just culinary masterpieces that are beautiful, but also something that will utterly delight our users.

2. BACKGROUND

Optimizely was formed in 2010 having completed a successful Y Combinator startup program. Optimizely offered easy setup for A/B testing. Simply by inserting a single line of JavaScript, someone could start running A/B tests on their website. By 2013, having grown to approximately twenty engineers, product delivery had stalled due to increased communication bandwidth and too much work in flight. In 2014, the teams learned about Scrum to improve product delivery and brought on Keith Nottonson to help them on their agile journey.

By the close of 2015, the product, design and engineering teams had been running two-week sprints for eighteen months, greatly improving product delivery but still something was off. Though we had solved our original problem of product velocity, people were beginning to tire of the sprints themselves. We experienced local optimization as teams were working off their own backlog, often on work that was lower priority than items on other teams' backlogs. Sprint reviews were poorly attended and retrospectives had become a chore. We were not innovating enough and there was not enough time spent on discovery, customer development and prototyping. Something needed to change.

Dae-Ho Chung, Optimizely; email: tsukino@gmail.com
Keith Nottonson, Optimizely email: keithn@gmail.com
Jeffrey Sing, Optimizely; email: jeffsing@gmail.com

At the end of 2015, Marty Cagan came and did a workshop with Optimizely where we learned about dual-track Scrum. From there, we proceeded to create a discovery Kanban system. As described at Agile 2019 in Enterprise Service Planning at Optimizely [1], we eventually combined the discovery system with our delivery system to create an end-to-end customer Kanban.

Now in 2020, we have more than a dozen teams working on ideas that flow through this end-to-end process, and we use innovation and experimentation throughout our product development process.

3. INNOVATION AT OPTIMIZELY

We continually strive to build a culture that enables us to build revolutionary new things and not just evolutionary improvements. To that end, we nurture innovation in a variety of ways at Optimizely. The theory behind our innovation model is based on Geoffrey Moore's Using Innovation to Thrive and Strive [2]. Geoffrey Moore writes "[t]he idea behind the model is that established enterprises embracing disruptive innovations need to organize around four zones, each with its own operating model, each with well defined APIs to the other three zones." Many people focus on innovation that is disruptive, like the creation of a wholly new product or business model, which typically takes place in what Geoffrey Moore calls the Incubation Zone. "This is where all the futuristic initiatives live, ones that are not expected to pay back in any foreseeable future, but which are intended to create options to participate in the next-generation disruptive innovations."

However, there are other types of innovation, including performance and productivity innovations that take place in the other quadrants, and we encourage these types of innovation as well as disruptive ones. One way we encourage all the different types of innovation is with our twice-yearly hackathons.

Hackathons are organized time boxes, typically one day to one week, where people can explore ideas outside of their daily work streams. Upon joining Optimizely, Keith Nottonson became the facilitator and host of the hackathons with Dae-Ho Chung now leading them. Hackathons are a good opportunity for people to build out ideas in code, learn a new skill, or investigate a new solution, architecture or language. Not all innovation is product based. Engineering practices have also evolved out of hackathons, including our increasing use of paired programming and our engineering blog. We have had hack weeks, hack days, and even a hack sprint. [3]

For the last several years, we have been running two hack weeks a year: one in the winter and one in the summer. These have consistently led to 30-plus hack presentations of which half a dozen in their original form usually make it to production, often in the form of productivity and performance innovations. Over the half a decade, dozens of hacks have become roadmap backlog items and been delivered to our customers.. We call this post-hack week phase of our process "Pursuing" and all the hacks are tracked as a program to make sure we do not waste an opportunity.

However, we also needed a way for the organization to allocate some dedicated time to exploring new markets and ideas. We accomplished this with our Horizon 3 process, which seeks to allocate upwards of 10% of our overall capacity to exploring new things.

The head of product along with our product and engineering leadership select several ideas from the pool of ideas, and equip the idea with a product manager, a product designer and an engineer, who spend approximately 50% of their time on exploring this idea for at least one month. Some of these ideas may be wide open like omni channel or machine learning or more specific like solving for single page applications for our customers.

Once you've tested some ideas and answered some questions, there will be a Persevere, Pivot, or Kill decision. [4] This is a monthly check in with the product and engineering leaders where the team reviews the questions and hypotheses they set out to answer, and what they learned from them. Each team may use a different tactic each month to learn what it can about the selected idea. Finally, they make a recommendation for how to proceed with this idea. After reviewing all the ideas each month, the head of product along with the product and engineering leaders decide how each idea should proceed, if at all, for the next month.

Though we may not always attain this percentage of time, some of the style and patterns learned during these Horizon 3 cycles stay with the team, including 'out of the box' meetings where a team will spend an hour every week or sprint, thinking about things not in their day to day. And then there are the twice yearly hack weeks where they can take their idea further if they choose.

4. EXPERIMENTATION DURING DISCOVERY

Jeff Zych, former Head of Design at Optimizely wrote, “In the Discovery phase, the goal is to understand and solve our customer’s problems. The output is finished designs that solve for all use cases.” [5] As ideas, including ones from hack week and Horizon 3, flow through the discovery process, we talk with our users. We seek out the right people to speak with by running painted door tests and we also can use butter bars and surveys to find appropriate users and conduct research.

Another way to experiment during the discovery process is to look for small wins and small, testable hypotheses. Don’t spend months building a prototype and try to boil the ocean and make everything work. Find ways of “faking” the experience or technology to rapidly prototype ideas. Aim to prototype and test an idea in 1-2 weeks. For example, at Optimizely, we often create a frontend experience that mocks certain functionality using JavaScript. This isn’t a fully built product, but may mock a certain workflow that our customers may be interested in. We can get this built pretty quickly and out facing our customers within the workday. We track the usage metrics and conversions to determine if our customers are interacting with our prototype in ways we thought they would. If they don’t, we can rapidly change parts to determine what was working vs. what does not. This helps us to quickly iterate the right functionality by backing what we built with actual usage metrics.

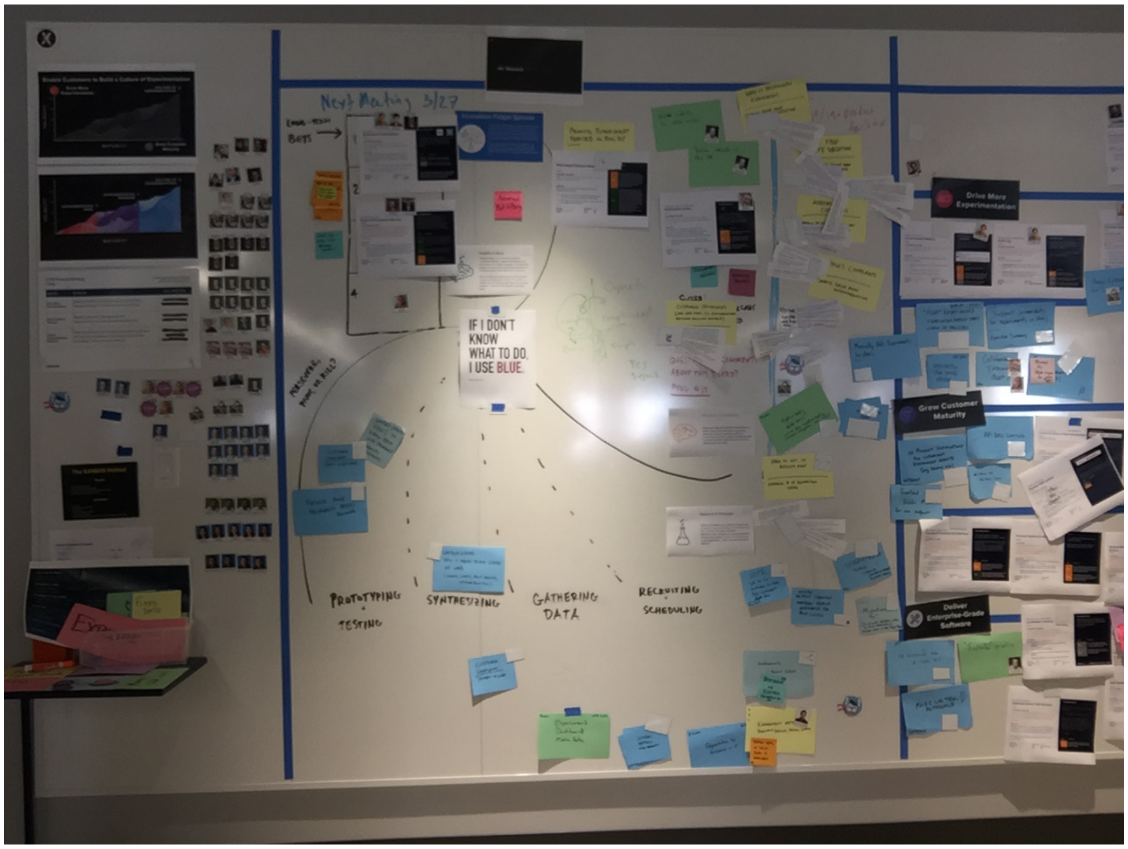

Figure 1. Horizon 3 Innovation Cycle as tracked on the Wall of Work May 2018 (Photo Keith Nottonson)

5. EXPERIMENTATION DURING DEVELOPMENT

Experimenting during development can help ensure you are building the product right. Two indicators that Optimizely uses are velocity (can we ship code faster?) while maintaining high quality (happy customers). By using Feature Flags, also known as Feature Toggles, Optimizely has implemented feature testing (does this work better/faster?), feature gating (do the right people see this?), rolling out in stages (does this scale?) and safe rollbacks (can I recover quickly?). These all ensure that our code gets shipped fast and our quality control stays high.

Feature testing allows us to run an A/B/n experiment on implementations of our features and allows us to validate that the implementation we end up shipping is indeed the best or fastest. We accomplish Feature Testing by using Feature Variables in Optimizely Full Stack. For each individual feature that we are testing, we can code different behaviors behind parameters. These parameters are modified by the feature flag. This allows us to test different values for these parameters without having to go back and change code (this makes feature testing extremely fast since we didn't need to have multiple builds with different code). For example, Optimizely ran a Feature Test in development against its Stats Accelerator algorithm. With any machine learning model, in order to get the most benefit you need to tune the hyper-parameters which control things like how quickly traffic is pushed to each variation and how to handle fluctuating conversion rates. For our variations in this feature test we tested an aggressive variation (aggression value higher) vs. a more passive variation (aggression value lower). We found that our aggressive variation was statistically significantly faster than the passive variation which gives us high confidence this is the best implementation.

Feature Flagging allows us to mitigate development risk by allowing us to test in production safely and quickly. Every week we deploy code at a certain cadence per week. Developers do not have to wait for the entire feature to be completed before they ship to production, since they can hide their incomplete code behind a feature flag. Once the feature is complete, our developers turn on the code only to a select group of audiences. For our Stats Accelerator project, we turned on this feature in production to our quality engineering (QE) team initially. This allowed for the QE team to test our features in the actual production environment in which our customers are using them. Then, we turned on Stats Accelerator to a select group of beta customers. This allows us to quickly get feedback and allows us to make fixes without exposing issues to the rest of our customers.

Rollout in stages allows us to slowly introduce a feature to our customers by selecting what percent of our customer traffic will have access to our new features. Since the code is already deployed, we can adjust more or less traffic depending on what we are monitoring (error rates or performance). This allows us to quickly validate engineering quality and our backend infrastructure's ability to scale and support the load. For our Stats Accelerator project we rolled out this feature to initially 20% of our customer base. After a week we validate that our infrastructure could handle the increase in traffic and computation requirements, and that no major issue was reported. We then rolled out to 50% /80%/100% with this process ensuring that the full deployment was seamless.

Even though our Stats Accelerator example was deployed without issues, we could have quickly mitigated any issues through a roll back. Rolling back the code is a simple on/off switch where we can deactivate the wayward feature. This significantly speeds up your Mean Time to Resolve. Without a feature flag, a bug that caused an incident would require teams to come together, sort through the entire deploy to find what caused the issue, debate over rolling back or fixing forward, and then redeploy the build with the fix. Since features released behind feature flags are independent of build changes, we know pretty quickly when a feature flag is the culprit behind the incident. By turning off the flag, the offending feature is immediately rolled back without having to redeploy any code. By using a feature rollback, Optimizely's average Mean Time To Resolve Incidents due to a new feature has dropped by over ~84% (Average hotfix time in 2019: ~ 2 hrs, Average rollback time: ~20 min).

6. EXPERIMENTATION AFTER LAUNCH

We have used experimentation to make sure that the product we built is something our customers truly want. We have used experimentation throughout development to prove that our product is working correctly. We have now confidently launched our product. However, the work is still not done. There are two big needs left to address:

- How do we measure the business impact so we can adjust our product iteratively; and
- How do we drive more adoption of our product so more customers use it?

Upon launch of our product, we seed certain events that we track in our new product. For our Stats Accelerator (SA) experiment, we tracked how many customers started a new SA experiment, and how many customers actually had an SA experiment reach statistical significance. These events give us indicators which we can measure regarding how many customers adopted our new product (started), and if customers were understanding the use case that our solution presented (reached statistical significance).

What we find occasionally is the products we launch may require more hand holding to understand the full potential of usage. Here, we can run a few experiments regarding help texts, starter guides, or adoption guides.

We track the usage or clicks of our help texts or guides, and see if they convert to more usage of our product. If they do, our experiment is successful and we have driven more business impact for our new product. If they do not, we make adjustments, and run a new experiment. Iteratively we continue until we find the right solution to bring us more lift.

For the question regarding how do we drive more adoption and customer usage, we often run targeted campaigns against a targeted market. For our SA example, we chose to run a Personalization campaign for accounts that have A/B Test but not SA enabled to drive upsell. We used a Personalization campaign that had messaging around how we could help our customers reach statistical significance faster using our new SA product, and links to try it free. We measured against conversions to see how many customers contacted us for free trials, and adjusted our experiments to see if we could drive higher signups.

Using experimentation in our launch stages put the final touches on your end product. It ensures that you are really getting the usage you expect your customers to get, and it allows you to figure out how to drive higher adoption.

7. WHAT WE LEARNED

Using an experiment-driven product development cycle helps identify and create better value earlier and more often. As Whelan Boyd writes " Leveraging experimentation in each of the four steps can help you avoid building the wrong product and building the product wrong." [6]

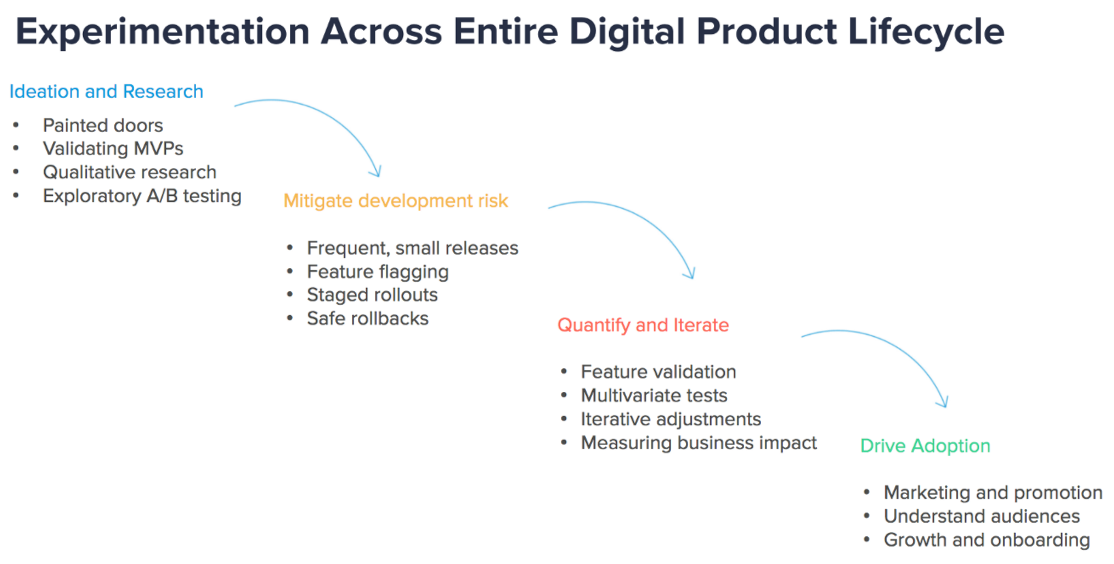

Figure 2. Experimentation across entire digital product lifecycle by Whelan Boyd

Building this culture of experimentation to drive innovation at Optimizely, we have discovered a few key concepts that we are still driving to improve. We also learned that experimentation must be done at every level in the company. From the product team building the roadmap which flows to the engineering team building the product to the marketing team trying to help customers find and use the product.

At the product level, the most important lesson was to show ROI. This helps tie business impact to the experiments run. By having a good ROI model, it is immediately apparent the more experiments run, the more business gains.

At the engineering level, we aggressively dog food our own product (the practice of an organization using its own product to test its validity). A lot of what we learned during our experimentation resulted in changes in features or best practices. We found in order to properly implement a platform (feature management system) that allows us to experiment in all stages, we needed to establish proper governance.

If the experimentation program was an airport, and our feature flags were airplanes, then we needed an air control tower (governance program) to properly orchestrate all the moving parts and to ensure the final product didn't end up in a fiery mess. This governance program was a mix of policy, process, and smarter automation. Things like how long a feature flag should live in your code, or what risk profiles do we need to assign to certain flag types were created and processes such as feature flag removal or run books were enacted. We also built and shipped features to aid in this process such as our Jira integration with Optimizely. This allowed us to internally automate things such as automatically creating a feature flag through Jira.

Finally, have the right experiment champions lead and be in charge of coaching the experimentation process. Your champion should be helping teams across domains to set up the right programs and process. No team's experimentation program or process will be identical to another, which is why the right metrics to measure and define success needs to be clearly measured and tracked. At Optimizely, we had a technical program manager that organized our experimentation program and measured the velocity of experiments created, and fostered best practices and aides to run them. Your experiment champions need to put in place the right governance process to ensure that the mechanics of running experimentation across the product life cycle is repeatable. Creating a bespoke experiment campaign per each feature will lead to low engagement over time. Having set standards and ways to enforce and guide them will make your experiment process more repeatable and successful.

8. ACKNOWLEDGEMENTS

We would like to thank Rebecca Wirfs-Brock and Frank Olsen for their patience, support and encouragement. We would also like to thank all our fellow Optinauts, past and present, who have gone on this journey with us. In particular, Whelan Boyd's article, "How to Avoid Building the Wrong Product? Make Your Product Roadmap Your Experimentation Roadmap," and Asa Scahchar's Product Tank presentation, "Experiment Led Product Management" were extremely informative."

REFERENCES

[1] Keith Nottonson, https://www.agilealliance.org/resources/experience-reports/enterprise-service-planning-at-optimizely/
[2] Geoffrey Moore, https://www.slideshare.net/sasindia/keynote-geoffrey-mooreusinginnovationtothriveandstrive
[3] Agile Rock, https://medium.com/engineers-optimizely/why-we-make-time-to-hack-bc3697c47d0
[4] David Bland, https://medium.com/@davidjbland/spruce-the-corporate-minimum-viable-product-fc251dc1a424
[5] Jeff Zych, https://jlzych.com/2016/07/17/discovery-kanban-at-optimizely/
[6] Whelan Boyd, https://medium.com/product-experimentation/how-to-avoid-building-the-wrong-product-make-your-product-roadmap-your-experimentation-roadmap-a2847babfeee